UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

# MUSEUS VIRTUAIS

Rui Pedro Pelica Santos Flores

PROJECTO

MESTRADO EM ENGENHARIA INFORMÁTICA
Sistemas de Informação

2012

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

# MUSEUS VIRTUAIS

Rui Pedro Pelica Santos Flores

PROJECTO

Trabalho orientado pela Professora Doutora Maria Beatriz Duarte Pereira do Carmo
e co-orientado pela Professora Doutora Ana Paula Boler Cláudio.

MESTRADO EM ENGENHARIA INFORMÁTICA
Sistemas de Informação

2012

# Agradecimentos

Este documento representa o culminar de um longo percurso na minha vida pessoal e intelectual, e como tal, um dos momentos mais importantes da minha vida.

Foi um longo percurso, com altos e baixos, mas é com muita alegria que atinjo, por fim, este objectivo da minha vida.

E por isso, tenho alguns agradecimentos a fazer. Primeiro, à minha família, os meus pais e irmão, especialmente, pois sem eles e a sua ajuda e apoio incondicionais, seria muito mais difícil percorrer este longo caminho, agradeço especialmente ao meu pai, que, a todos os níveis, nunca deixou de me apoiar, mesmo nas alturas mais difíceis.

Agradeço também à minha namorada, Marta Lourenço, a sua paciência para comigo e o facto de estar sempre ao meu lado para me animar, acalmar e aconselhar dentro e fora da Universidade, nos bons e maus momentos.

Aos meus amigos, de maior ou menor data, que são muito importantes na minha vida e que, mesmo sem saber, contribuiram para que este caminho não fosse tão duro. Agradeço especialmente a três grandes amigos, André Dias, Tânia Albuquerque e Tiago Rocha.

Neste longo percurso, fiz também boas amizades, e sem dúvida que não foram menos importantes. Por isso não podia esquecer de agradecer a João Lobo, Henrique Morais, Louis Lopes, Geraldo Nascimento, Gonçalo Cruchino, Frederico Miranda e Nuno Fonseca a sua ajuda e o seu companheirismo.

Aos colegas de investigação do LabMAg, pelos quais fui muito bem recebido e que contribuíram para um óptimo ambiente de trabalho, especialmente ao Nuno Henriques, Fernando Silva, Jorge Gomes, Edgar Montez João Silva e Davide Nunes.

Finalmente, falta-me agradecer à minha orientadora, Beatriz Carmo e co-orientadora, Ana Paula Cláudio, que deram um precioso contributo ao longo do projecto e na redacção deste relatório.

*À amizade e ao companheirismo.*

# Resumo

Para os museus e galerias de arte a divulgação das suas exposições através da Web é importante, quer para atrair visitantes, quer para a difusão de património artístico e cultural. Idealmente pretende-se que a experiência de visitar virtualmente uma exposição, através de modelos virtuais, seja a mais aproximada possível de uma visita ao espaço físico real.

O desenvolvimento de ferramentas para a criação de exposições virtuais, através de modelos tridimensionais, além de possibilitar a divulgação das exposições, permite ainda auxiliar a concepção e montagem da própria exposição. Existia já uma aplicação elaborada no âmbito da equipa de investigação em que este projecto se integra, Virtual Exhibition Builder, que recorre a modelos tridimensionais em X3D.

Existe agora uma maior aposta em aplicações Web baseadas em modelos tridimensionais, contribuindo, para isso, o recente *boom* das aplicações 3D na Web, através de novas tecnologias, como WebGL.

O principal objectivo deste projecto foi o desenvolvimento de uma ferramenta para auxiliar os técnicos de museus, normalmente sem experiência no domínio da informática, na fase inicial de concepção de uma exposição. Além disso, pretendia-se que as exposições construídas pudessem ser divulgadas através das páginas web dos museus.

Numa primeira fase fez-se um trabalho de pesquisa sobre a tecnologia WebGL. Concluiu-se que a adopção do WebGL não trazia vantagens relativamente à tecnologia usada na aplicação já desenvolvida.

Optou-se, assim, por extender a aplicação já existente, juntando novas funcionalidades, de que se destacam, a visualização de valores das distâncias de uma obra em relação a pontos de referência., a adição de vitrines, a criação de representações de obras 3D com baixo nível de detalhe e a criação de uma aplicação interactiva para criação e actualização de uma base de dados de obras de arte.

**Palavras-chave:** Exposições, Virtual, X3D, WebGL

# Abstract

For museums and art galleries, the disclosure of their exposure through the Web is important for attracting visitors and the diffusion of artistic and cultural patrimony.

Ideally it is intended that the experience of visiting a virtual exhibition using virtual models is the most close as possible to a real visit on the physical space.

The development of tools for the creation of three-dimensional virtual exhibitions allows the disclosure of those exhibitions and helps in their design.

There was already an application developed by the research team that proposed this project, the Virtual Exhibition Builder, that uses X3D tridimensional models.

There is now a big investment on Web applications based in three-dimensional models, contributing to this, the recent boom of 3D applications on the Web, through new technologies such as WebGL.

The main objective of this project was the development of a tool to support the team of a museum, who usually are not experts in informatics, in the initial design of an exhibition. Moreover, it was intended that the exhibitions could be disseminated through the web pages of the museums.

The first stage of this project was to study the WebGL technology.

We concluded that the adoption of this technology would not bring any advantages if compared with the application already developed, so we decided to extend the application Virtual Exhibition Builder adding new functionalities, such as, the visualization of the distances of an artwork on its surface based on reference points, the adding of vitrines, low level detail representations of tridimensional artworks and the implementation of an interactive application for the creation and updating of artworks databases.

**Keywords:** Exhibition, Virtual, X3D, WebGL

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1 Introdução

Este capítulo tem como objectivo a apresentação da motivação, dos objectivos do trabalho desenvolvido, as suas contribuições e finalmente a organização e estrutura do documento.

## 1.1 Motivação

A divulgação do acervo de museus e de galerias de arte através da Web é de grande relevância para as instituições. Além do contributo cultural em termos da divulgação de obras de arte, atingindo um público mais alargado do que aquele que habitualmente frequenta as suas instalações, a disseminação do acervo de um museu é também um meio para captar potenciais visitantes e um auxiliar para que estes possam tirar maior partido da sua visita. Por outro lado, é uma forma de dar visibilidade a obras que temporariamente não estão expostas, seja devido a restrições de espaço, seja pela sua fragilidade, seja por motivo de acções de restauro em curso. Para lá da colecção do próprio museu, através da Web podem ainda ser divulgadas exposições temporárias, que mostram a dinâmica cultural de um museu.

Se a tudo isto, estiver associada uma ferramenta que permita que as pessoas possam navegar livremente por um espaço virtual tridimensional, naturalmente que a experiência se tornará mais enriquecedora, em comparação com as ferramentas existentes, baseadas em fotografias panorâmicas, como em [visVirt3D].

O desenvolvimento de ferramentas para a criação de exposições virtuais, através de modelos tridimensionais, além de possibilitar a divulgação das exposições, permite ainda auxiliar a concepção e montagem da própria exposição.

No âmbito da criação de uma ferramenta para auxílio de equipas de museus na elaboração de exposições, foi criada a Galeria de Arte Virtual [Semião08]. Esta aplicação foi reformulada e estendida dando origem à Virtual Exhibiton Builder [Gomes11]. A aplicação Virtual Exhibition Builder foi apresentada à equipa de um museu, motivando a proposta deste projecto que visou a criação de uma ferramenta mais

robusta, com mais funcionalidades e explorando novas tecnologias, com destaque para o WebGL.

A aplicação Virtual Exhibition Builder tinha algumas limitações, quer ao nível das funcionalidades desenvolvidas, quer ao nível de realismo das imagens, entre estas, a projecção de sombras. A solução deste problema poderia passar pela utilização de novas ferramentas, que concretizassem tudo o que estava desenvolvido e pudessem resolver as limitações apresentadas.

## 1.2 Objectivos e contribuições do trabalho

Este trabalho foi realizado no âmbito da Projecto em Engenharia Informática (PEI), para o Mestrado em Engenharia Informática, na Faculdade de Ciências da Universidade de Lisboa, num dos grupos de investigação do Departamento de Informática, o LabMAg – Laboratório de Modelação de Agentes.

O principal objectivo da elaboração deste projecto foi construir uma aplicação que auxilie os técnicos de museus a construir uma exposição, tendo também em linha de conta, permitir "visitas virtuais" à exposição. Este projecto deu seguimento aos trabalhos Galeria de Arte Virtual e Virtual Exhibition Builder, desenvolvidas por Pedro Semião e Jorge Gomes, respectivamente. Estes trabalhos basearam-se em modelos tridimensionais elaborados em X3D.

Este projecto visou a criação de uma ferramenta mais robusta, com mais funcionalidades. Estas foram sugeridas pela equipa do Museu da Cidade da Câmara Municipal de Lisboa à qual foi mostrada a ferramenta Virtual Exhibition Builder [Gomes11], e de que se destacam as seguintes: visualização da informação das distâncias das obras nas paredes, criação de uma interface mais amigável para criação e actualização de bases de dados, criação de vitrines como novo tipo de objectos amovíveis, possibilidade de associar várias imagens a um objecto de forma a construir um modelo tridimensional com baixo nível de detalhe.

Atentendo às limitações no realismo da aplicação existente, em particular, no que diz respeito à projecção de sombras, explorou-se outra tecnologia para a construção da aplicação: o WebGL. Esta tecnologia foi explorada de modo a verificar a viabilidade da elaboração de uma aplicação de raíz baseada nas funcionalidades já desenvolvidas na aplicação Virtual Exhibition Builder. Para isso, foi desenvolvido um protótipo que serviu de treino no uso da tecnologia WebGL e no qual foram definidas várias tarefas, ou *checkpoints*, baseadas nas funcionalidades já existentes na aplicação Virtual

Exhibition Builder. Verificou-se que a tecnologia WebGL permite implementar aspectos que não foram possíveis de conceber, particularmente, a projecção de sombras através de fontes de luz, que foi um dos principais motivos que levaram à exploração desta tecnologia. Contudo os resultados não demonstraram o efeito esperado, isto é, existiam inconsistências nas projecções de sombras.

Face aos resultados obtidos entre a extensão da aplicação Virtual Exhibition Builder e a concepção de uma ferramenta equivalente, mas com mais funcionalidades, recorrendo à tecnologia WebGL, escolhemos a primeira opção: estender a aplicação Virtual Exhibition Builder concretizando as funcionalidades sugeridas pela equipa do Museu da Cidade da Câmara Municipal de Lisboa.

## 1.3 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 1 – Introdução

  Este capítulo descreve a motivação, objectivos e a estrutura do documento.

- Capítulo 2 – Trabalho Relacionado

  Este capítulo descreve trabalhos realizados na área em que este projecto se insere.

- Capítulo 3 – Trabalho Desenvolvido – WebGL

  Este capítulo descreve os ensaios realizados com o WebGL.

- Capítulo 4 – Trabalho Desenvolvido – X3D

  Este capítulo descreve o que foi desenvolvido tendo por base a organização da cena em X3D.

- Apêdice A – Manual de Utilização da Aplicação Java/X3D
- Apêndice B – Manual de Utilização da Aplicação WebGL
- Apêndice C – Diagrama de Classes
- Apêndice D – Mapa de Gantt
- Bibliografia

# Capítulo 2 Trabalho Relacionado

Este capítulo refere-se a diversos trabalhos desenvolvidos por outro autores na área em que este projecto está inserido.

Descrevem-se em seguida esses trabalhos:

## 2.1 Virtual Art Gallery

A aplicação Virtual Art Gallery [Hrk01] tem um editor gráfico 2D que permite, por um lado, a edição da planta do espaço de exposição, alterando, por exemplo, a cor ou textura associada a uma dada parede, e por outro lado, a colocação de quadros nas paredes. Esta segunda tarefa é realizada usando duas janelas auxiliares: uma com a visualização das áreas cobertas pelos quadros colocados, outra com a interface para escolha da obra e indicação das coordenadas para colocação do quadro. A inserção interactiva de objectos tridimensionais não é tratada.

## 2.2 ARCO

No âmbito do projecto ARCO (Augmented Representation of Cultural Objects) desenvolveram-se um conjunto de ferramentas destinadas a construir e gerir exposições virtuais de museus [Woj04]. Neste conjunto incluem-se ferramentas para o apoio à produção de modelos digitais de obras de arte, à gestão de um repositório, onde podem ser guardados vários tipos de representações para estas obras, e à sua visualização num espaço tridimensional. A criação de exposições virtuais pode ser feita através de um conjunto de "templates" seleccionando os parâmetros adequados e os modelos VRML/X3D guardados no repositório. Estes "templates", que definem tanto o aspecto visual como o comportamento da apresentação, são construídos em X-VRML, uma linguagem de alto nível baseada em XML e que estende o VRML (Virtual Reality Modeling Language).

## 2.3 The Development of an E-Museum for Contemporary Arts

Outro exemplo de criação de um museu virtual é descrito em [Patias08]. O trabalho desenvolvido teve por objectivo a divulgação, através da Web, do Museu de Arte Contemporânea da Macedónia, em Tessalónica, na Grécia. O espaço físico do museu e as obras (que são digitalizadas para uso na aplicação) são convertidos num formato Web3D (VRML/X3D) para poderem ser visualizados na Web. Uma das funcionalidades disponibilizadas aos utilizadores é a possibilidade de construção interactiva de uma galeria de arte. Os objectos de arte podem ser seleccionados através de uma base de dados de obras de arte baseadas em palavras-chave, como o nome do autor ou da obra, entre outros. Os objectos são depois colocados por "arrastamento" no espaço virtual. Não são indicados detalhes sobre a aplicação.

## 2.4 Browserbased 3D Gallery Builder

A ferramenta Browserbased 3D Gallery Builderm, que foi elaborada por um grupo denominado Katalabs [Katalabs], permite a construção de exposições através de um Web browser, recorrendo à tecnologia WebGL, utilizando a biblioteca GLGE . Esta ferramenta permite a criação de exposições virtuais tridimensionais, oferecendo a possibilidade de colocação de obras de arte 2D (quadros) de uma forma bastante simples, indicando apenas o URL de uma qualquer imagem da Web. Também é oferecida a possibilidade de colocação de obras 3D, como esculturas, não existe contudo informação de como estas são introduzidas. Não são fornecidas mais informações sobre a ferramenta, no entanto pode-se visualizar uma demonstração da ferramenta em [galBuilder].

## 2.5 Galeria de Arte Virtual

A aplicação Galeria de Arte Virtual [Semião08], foi desenvolvida no âmbito de um projecto de Engenharia Informática em colaboração com o Instituto Açoriano de Cultura.

Foram considerados alguns requisitos para esta aplicação, nomeadamente a facilidade de utilização por pessoas sem experiência informática e a possibilidade de reutilização para vários espaços expositivos. A opção recaiu em suportar os modelos virtuais em

X3D. A manipulação da cena é feita com recurso ao Xj3D e a informação sobre os quadros está guardada numa base de dados MySQL. A construção da exposição foi dividida em duas fases: na primeira, são identificadas as superfícies onde se desejam colocar os quadros, criando um novo ficheiro com sensores de toque associados às superfícies seleccionadas. Na segunda fase, constrói-se a exposição virtual pesquisando os quadros na base de dados, e seleccionando as superfícies onde devem ser colocados [Semião08].

Entre as limitações detectadas nesta ferramenta destacaram-se: o tratamento de apenas algumas das representações possíveis no formato X3D, a impossibilidade de alterar uma exposição criada numa edição anterior e a colocação apenas de obras de arte bidimensionais, como quadros e tapeçarias. Para colmatar estas limitações foi desenvolvida uma nova aplicação, mais genérica e com mais funcionalidades, que será apresentada a seguir.

## 2.6 Virtual Exhibition Builder

A aplicação Virtual Exhibition Builder [Gomes11] estendeu a aplicação descrita acima (Galeria de Arte Virtual), tendo como objectivo complementar as funcionalidades dessa aplicação.

Relativamente à aplicação Galeria de Arte Virtual, foram feitas algumas melhorias, nomeadamente a conversão da descrição dos modelos tridimensionais numa geometria que possibilite um tratamento uniforma de cenários, a possibilidade de edição de exposições já construídas; a criação de filtros auxiliares para identificação de superfícies elegíveis para colocação de obras de arte; a extensão do tipo de obras de arte que podem ser colocadas no espaço expositivo, nomeadamente, a possibilidade de inclusão de objectos 3D; a inserção de objectos auxiliares para suporte à apresentação de obras de arte, por exemplo, divisórias amovíveis e bases para colocação de esculturas e a possibilidade de definir viewpoints.

Vamos agora focar-nos numa descrição mais detalhada desta aplicação, dado que esta será estendida neste nosso projecto e é de toda a conveniência deixar claro o que já estava feito e o que nós fizemos.

A seguir explicamos com mais detalhe esta aplicação.

### 2.6.1 Arquitectura da aplicação

Nesta secção é descrita a arquitectura da aplicação Virtual Exhibition Builder.

A figura 2.1 apresenta o diagrama de classes simplificado com os componentes da aplicação e a sua articulação:

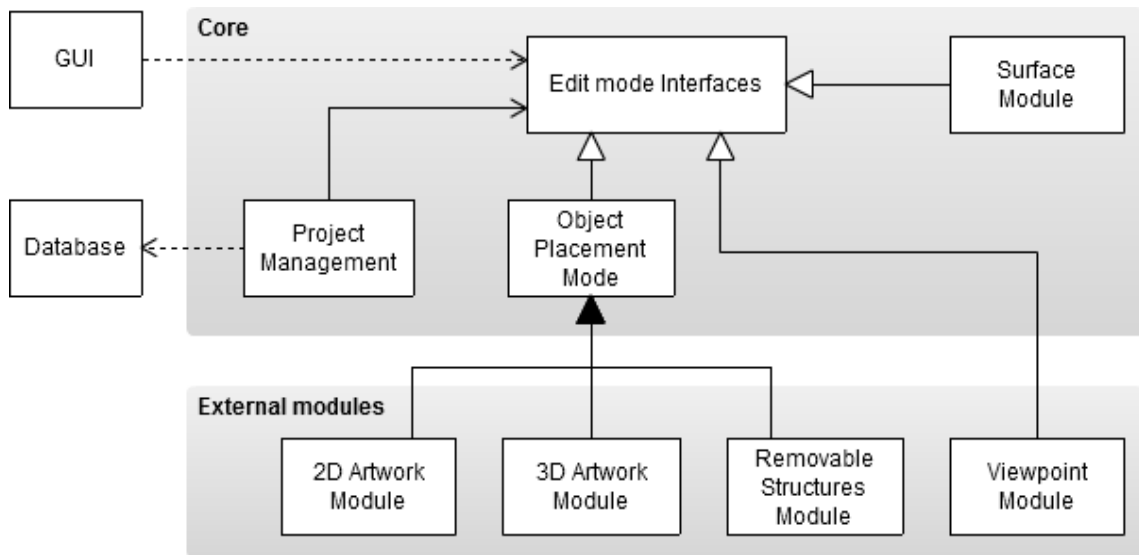



**Figura 2.1 - Diagrama de classes simplificado [RTGomes11].**

A arquitectura da aplicação baseou-se num design modular. Este aspecto tem particular relevância se considerarmos que a criação de uma exposição nesta aplicação envolve vários processos e tipos de objectos, que se querem independentes.

De forma a oferecer esta modularidade, as funcionalidades da aplicação foram separadas em vários módulos, unidades independentes associadas a uma tarefa específica, como seleccionar paredes ou inserir obras 2D ou 3D (Figura2.1).

A articulação entre os módulos foi assegurada recorrendo a interfaces bem definidas que contêm os métodos necessários a uma boa fluidez na aplicação, tal como as transições de módulos, por exemplo.

Os módulos têm dois estados: activo e inactivo. São possíveis transições de módulos a qualquer momento, sendo o último estado de cada módulo guardado, para que, regressando a ele, o utilizador possa continuar o trabalho feito até então.

Dado que várias tarefas da aplicação se relacionam com inserção de objectos na cena, foi criada uma implementação abstracta referente colocação de objectos. Esta implementa todas as tarefas comuns aos vários tipos de objectos colocados na cena, como o próprio acto de colocação, deslocações e o desenho das *bounding boxes*. Para cada tipo específico de objectos, um módulo específico foi implementado, derivado do

módulo abstracto. Cada módulo descreve as características particulares de cada tipo de objecto. Este módulo abstracto teve por fim facilitar a implementação e introdução de novos tipos de objectos na cena.

## Software utilizado

A informação das obras foi armazenada através de uma base de dados integrada na aplicação, uma base de dados SQLite. A opção por uma base de dados integrada evita instalação e configuração de servidores e bases de dados.

As inserções e edições na base de dados podem ser feitas com qualquer ferramenta compatível com bases de dados SQLite. Neste caso, a ferramenta utilizada para essas operações foi o SQLite Studio, que tem uma interface gráfica para esse fim.

A aplicação foi implementada sobre o Java Standard Edition 1.6 e os objectos X3D usam uma API Xj3D, versão 2.0M1, que constrói e manipula o grafo da cena. Para aceder e modificar este grafo foram usados os métodos do SAI (Scene Access Interface), parte da especificação do X3D.

A interface gráfica foi desenvolvida em Java Swing.

## Workflow da criação de uma exposição virtual

Descreve-se o workflow (Fig 2.2) para a criação de uma exposição usando a aplicação Virtual Exhibition Builder.

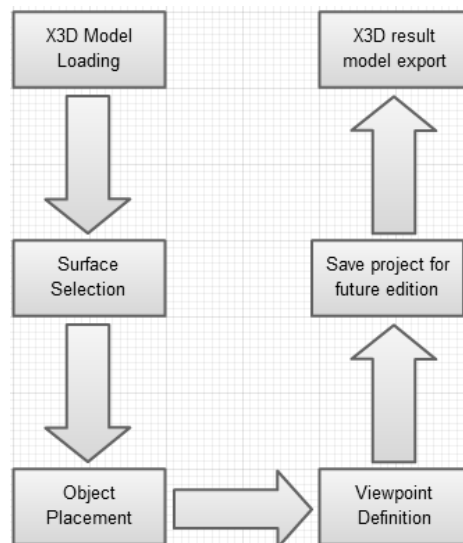



**Figura 2.2 - Workflow da aplicação.**

Para criar uma exposição, o utilizador é convidado a atribuir-lhe um nome e um modelo tridimensional X3D (Museu/Galeria de Arte), que irá representar o espaço físico da exposição. De seguida, são apresentados os módulos da aplicação: *Surface Selection, 2D Artwork, 3D Artwork, Blocks* e *Viewpoints*, podendo o utilizador começar a "montar" a exposição com as obras disponíveis na base de dados. Uma única base de dados foi construída e preenchida, denominada *sqlitedb*.

Ao carregar um ficheiro X3D com o modelo tridimensional do espaço físico da exposição é feita a uniformização da geometria da cena. Este passo é necessário porque, apesar do X3D ser um formato bem definido, existem múltiplas representações para o mesmo aspecto gráfico de uma cena dada a vasta variedade de nós que descrevem a geometria dos objectos.

Para resolver este problema, foi criado um processo que em nada altera a geometria inicial da cena, mas adiciona uma nova definição da geometria da cena. Esta camada abstracta da cena original é composta por superfícies invisíveis colocadas por cima da geometria original. Cada superfície é uma área definida por um vector normal e um conjunto de triângulos adjacentes com as normais paralelas. Todas as alterações operadas na cena são sobre esta camada abstracta, deixando a cena original intacta.

Antes de descrever os módulos e o processo de construção de uma exposição usando a Virtual Exhibition Builder, queremos salientar que o utilizador apenas pode executar tarefas num módulo de cada vez, ou seja, o processo de montagem de uma exposição envolve uma sequência de tarefas e cada tarefa é executada no seu módulo.

### 2.6.2 Módulos

Nesta secção descrevemos os módulos existentes na aplicação Virtual Exhibition Builder e o seu funcionamento. Para ilustrar melhor o que vamos descrever a seguir, usamos a figura 2.3. Esta figura apresenta a interface da aplicação. Para todos os módulos existe um botão na barra superior para os activar. Quando são activados abrem-se dois painéis na área de desenho: um do lado direito e outro lado esquerdo.

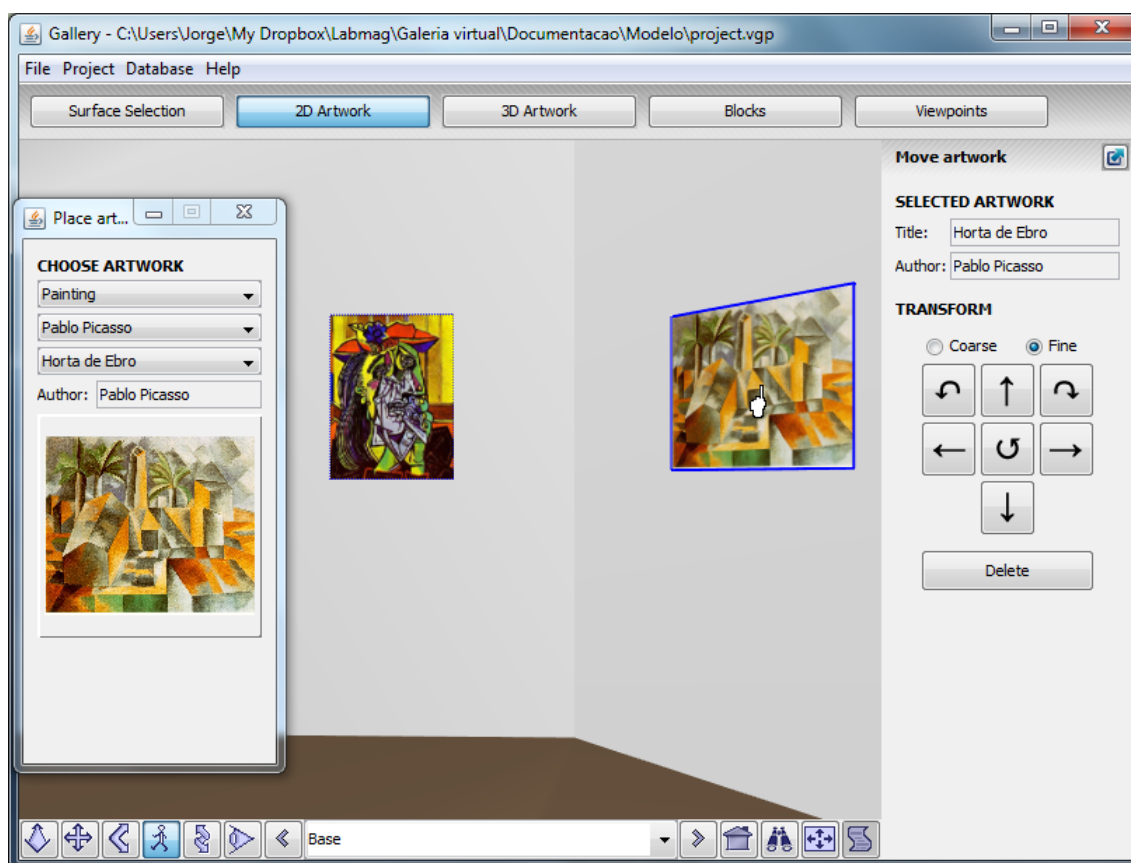



**Figura 2.3 – Interface geral da aplicação [RTGomes11]**

## Selecção de superfícies

Para começar a montar a exposição, o primeiro passo é a selecção das superfícies onde as obras serão colocadas, usando o módulo *Select Surface*. Para fazer isso, o utilizador tem duas opções: escolher manualmente, com um clique na parede desejada, interpretado através de um sensor de toque associado a cada superfície. Ou, alternativamente, o utilizador pode seleccionar automaticamente as superfícies recorrendo aos filtros desenvolvidos por área e posicionamento, por exemplo, seleccionar apenas paredes verticais ou seleccionar superfícies maiores que x metros quadrados. As superfícies seleccionadas terão uma côr distinta das demais (quando o *Select Surface* estiver activo).

## Obras 2D e 3D

O passo seguinte é colocar as obras nas paredes, sejam as obras bi ou tridimensionais.

Para a colocação de objectos na cena foi definida uma interface com os parâmetros básicos para a colocação de obras em superícies, nomeadamente, a criação da sua *bounding box*, a definição da superfície de contacto, o vector normal da superfície e a orientação do topo da obra.

Para a selecção da obra que o utilizador quer colocar na parede, existe um painel do lado esquerdo (Fig. 2.3) que permite ao utilizador pesquisar as obras que estão na base de dados, podendo filtrá-las por autor e tipo no módulo das obras bidimensionais, e apenas por autor nas obras tridimensionais, dado que não existe a noção de tipo de obra deste tipo, sendo todas consideradas esculturas.

A colocação de uma obra numa parede é feita através de um simples clique no lugar onde o utilizador quiser que a obra fique. Para deslocar a obra o utilizador terá de usar os botões de deslocação situados no painel do lado direito do módulo actual. Note-se que a colocação de obras bi e tridimensionais estão separadas em módulos distintos.

Falando das obras tridimensionais, existem duas representações: objectos detalhados construídos em X3D e objectos pouco detalhados, com a forma de um paralelepípedo, com uma imagem aplicada em todas as faces.

### Divisões

Também é possível inserir divisões no modelo (espaço físico da exposição) que podem, por exemplo, servir de apoio para esculturas ou mesmo representar paredes temporárias.

O método de inserção destas divisões é similar ao processo de colocação de obras, com a diferença de não ser, obviamente, necessário escolher uma obra, sendo apenas necessário definir as dimensões e a cor da divisão. Para definir essas informações está um painel do lado esquerdo com os campos para o utilizador usar, e no painel do lado direito o utilizador poderá, tal como nas obras bi e tridimensionais, usar os botões de deslocação.

### Viewpoints

Depois da exposição montada, isto é, de todos os objectos colocados nos devidos sítios, o utilizador pode, se desejar, definir viewpoints. Para isso, existe um módulo específico para criá-los e navegar sobre eles. Para definir um viewpoint é necessário o utilizador posicionar a câmara no lugar desejado e, no painel do lado esquerdo, atribuir-lhe um nome. Após a criação do viewpoint, este é adicionado a uma lista, visível no mesmo painel, que poderá ser utilizada posteriormente para navegar nos pontos já definidos.

No painel do lado direito o utilizador poderá alterar a localização de um viewpoint, seleccionando-o na lista já mencionada atrás, tal como eliminá-lo.

O utilizador poderá guardar a exposição para uma eventual futura edição, isto é feito através do registo do último estado de cada módulo. Este registo é efectuado num ficheiro, denominado ficheiro de estado, que contém a última configuração de cada módulo.

Por fim, o utilizador pode exportar a exposição para um ficheiro X3D.

Este ficheiro é criado e construído através de um processo em que cada módulo escreve as suas alterações num ficheiro de *output*. O ficheiro que contém o grafo da cena original é alterado utilizando a interface do DOM (Document Object Model) oferecida pela API do Java para processamento XML.

# Sumário

Neste capítulo apresentámos trabalhos de outros autores o âmbito de exposições virtuais. Descreveu-se com maior detalhe a aplicação Virtual Exhibition Builder [Gomes11] que serviu de base ao trabalho realizado neste projecto.

# Capítulo 3 Trabalho Desenvolvido - WebGL

Em virtude de haver algumas limitações na obtenção de imagens realistas em cenários baseado em X3D, testou-se a utilização de WebGL para averiguar se seria possível obter melhores resultados.

Inicialmente, para que pudéssemos testar o que era possível fazer com a tecnologia WebGL, começámos por pesquisar a sua origem, a sua relação com o OpenGL, a sua compatibilidade com os *browsers* da Web e as bibliotecas usadas.

## 3.1 O WebGL

O WebGL é uma biblioteca que estende a capacidade da linguagem JavaScript de gerar de ambientes gráficos interactivos 3D na web. O código WebGL é acelerado por hardware, isto é, usa directamente a placa gráfica do computador. Não tem necessidade de recorrer a plugins, podendo ser executado com qualquer browser compatível.

Este padrão é baseado no OpenGL ES 2.0 e fornece uma interface de programação de gráficos 3D. Ele usa o elemento Canvas do HTML5 e é acedido por meio de interfaces DOM (Document Object Model). A gestão automática de memória é fornecida como parte da linguagem JavaScript.

A especificação foi lançada, sob versão 1.0, em 10 de Fevereiro de 2011. O WebGL é administrado pelo Khronos Group.

## 3.2 O WebGL e o OpenGL

O WebGL é uma API DOM para criação de conteúdo 3D num browser. Baseada no OpenGL ES 2.0, o WebGL usa a linguagem OpenGL, GLSL. Além disso, dado que é totalmente integrado no browser, uma aplicação WebGL pode aproveitar a infra-estrutura do JavaScript Document Object Model (DOM) [webGLOpenGL].

## 3.3 Compatibilidade com os Web Browsers

Foi feita uma pesquisa acerca da compatibilidade do WebGL com os diferentes e mais usados browsers da web (Internet Explorer, Mozilla Firefox, Google Chrome e Safari) [wikiWebGL].

Esta era uma questão importante para percebermos se a nossa aplicação poderia correr no maior número de *browsers* possível, de modo a qualquer utilizador, em qualquer lado, a pudesse utilizar.

Quanto ao Safari, corre em Mac na versão 5.1, tendo de activar a flag Develop->Enable WebGL no Develop Menu. Relativamente ao Google Chrome, suporta nativamente WebGL a partir da versão 9 no Windows, Linux e Mac. O Mozilla Firefox suporta nativamente WebGL a partir da versão 4.

O Internet Explorer não suporta WebGL.

## 3.4 Bibliotecas WebGL

O passo seguinte foi efectuar uma pesquisa de bibliotecas WebGL de modo a facilitar o eventual desenvolvimento da aplicação.

Foram verificadas e experimentadas algumas bibliotecas, mais especificamente a SpiderGL [SpiderGL], GLGE [GLGE] , DAT.GUI [DAT.GUI] e a THREE [Three].

Optámos por usar a biblioteca THREE e a DAT.GUI, por observação e experimentação dos exemplos disponíveis pela biblioteca. Entre eles foi possível verificar alguns modelos tridimensionais desenvolvidos e a forma como eram feitas as projecções de sombras, que foi um dos fortes motivos pelo qual decidimos experimentar esta tecnologia.

Para a criação de interfaces para a aplicação, foi encontrada a DAT.GUI. De entre os recursos que foram consultados observámos que existiam algumas interfaces que poderiam ser úteis para a interacção do utilizador com a cena, o que envolve, por exemplo, a deslocação de obras nas paredes.

## 3.5 Ambiente de Desenvolvimento

O protótipo que será descrito na secção seguinte foi desenvolvido em ambiente Windows 7, recorrendo ao IDE (Integrated Development Environment) Netbeans Versão 7.0.1 [NetBeans] com a extensão para desenvolvimento de software PHP (que também serve para programar JavaScript).

Para criar o projecto basta ir ao Menu do Netbeans e seleccionar File->New Project, de seguida selecciona-se a categoria PHP com projecto PHP application.

Para armazenar os objectos referentes ao protótipo em si, nomeadamente as imagens, recorreu-se ao **XAMPP** [Xampp]**,** um servidor independente de plataforma, software livre, que permite o uso de uma base de dados MySQL, o servidor web Apache e os interpretadores para linguagens de script, como o PHP.

Para executar a aplicação, primeiro tem de se abrir o XAMPP, para isso terá de se aceder ao XAMPP Control Panel e executar os módulos Apache e MySQL.

Depois disto feito, terá de se ir ao Netbeans e configurar o *Run*, para o fazer: Run->Set Project Configuration->Customize. Na janela que será aberta acede-se ao Run Configuration (no menu à esquerda) criando-se uma nova configuração em "New", atribuindo-lhe um nome. Depois, no Run As, escolhe-se a opção "Local Web Site" e no Project URL e Index File refere-se a pasta e o ficheiro da aplicação, respectivamente. No caso deste projecto: "http://localhost/Testes/" e "PrototipoVersaoBeta003.html".

## 3.6 Protótipo WebGL

Nesta fase do projecto, optámos pela elaboração de um protótipo em WebGL que explorasse ao máximo as capacidades que se desejam ver na aplicação. A ideia foi a implementação de diversas pequenas tarefas já concretizadas manipulando um cenário em X3D, padrão usado para elaborar o espaço físico do museu nas aplicações às quais esta dá seguimento ([Semião08] e [Gomes11]) com o objectivo de perceber se a tecnologia WebGL consegue fazer tudo o que o X3D permite.

Todas as tarefas aqui descritas são baseadas na utilização da biblioteca THREE.js, à excepção da tarefa "interfaces de interacção com objectos", na qual será usada a biblioteca DAT.GUI.js.

Antes de começar por desenhar alguma coisa no viewport, tem de se criar uma “cena” e uma câmara. Estando a cena criada, todos os objectos criados têm de ser adicionados a essa cena. Outro aspecto importante é conferir-lhe iluminação. Finalmente, é necessário criar um renderer de forma a desenhar toda a cena no ecrã.

```
// criação da cena.
var scene = new THREE.Scene();
// criação da camara.
var camera = new THREE.PerspectiveCamera(FOV, ViewAspectRatio, Znear, Zfar);
scene.add (camera);
```

Sendo:

fov – Especifica o ângulo de visão, em graus, no eixo y.

ViewAspectRatio – Especifica a relação de aspecto que determina o campo de visão na direcção x. A relação de aspecto é a relação entre x (largura) para y (altura).

Znear – Especifica a distância entre o espectador e o plano de recorte mais próximo em z.

Zfar – Especifica a distância entre o espectador e o plano de recorte mais distante em z.

```
// criação de uma luz ambiente, dada a cor em hexadecimal (neste caso, luz branca).
var sun = new THREE.AmbientLight( color );
// Introdução do objecto sun na cena
scene.add( sun );
```

Sendo o argumento color um número hexadecimal.

```
// Criação de um renderer, do tipo WebGLRenderer, definindo também a sua dimensão.

var renderer = new THREE.WebGLRenderer( { antialias: true } );

renderer.setSize(Width, Height);

var container = document.getElementById( 'container' );

container.appendChild( renderer.domElement );

// Desenhar a cena a partir da câmara.

renderer.render( scene, camera );
```

### 3.6.1 Primitivas gráficas existentes

Para a montagem do espaço tridimensional da cena, é necessário conhecer as primitivas gráficas que a biblioteca oferece, de modo a facilitar essa montagem.

Recorrendo a esta biblioteca [Three.js], podemos facilmente construir cubos, esferas ou planos, sendo apenas necessário passar por argumento as dimensões desejadas.

Pensámos em contruir as paredes do espaço tridimensional recorrendo a paralelepípedos, tal como está feito nas aplicações a que esta dá seguimento.

Para o caso específico da criação de um paralelepípedo, basta o seguinte código:

```
// Criação do cubo com as dimensões desejadas, adicionando-o na cena posteriormente.

var cube = new THREE.CubeGeometry( sx, sy, sz );

scene.add (cube);
```

Sendo,

sx = Largura do cubo;

sy = Altura do cubo;

sz = Profundidade do cubo;

No protótipo, optámos por usar esta abordagem para desenhar as várias paredes no exemplo de modelo tridimensional elaborado. Posto isto, foi necessário, obviamente, definir o posicionamento das paredes. Para isso basta definir da seguinte maneira:

```
cube.position = new Vector3 ( px, py, pz );
```

Sendo px, py e pz as posições e x,y,z respectivamente na cena.

Também é possível atribuir cor aos objectos, bastando para isso criar um “material” com a cor desejada.

Para facilitar e criar alguma modularidade na codificação, optámos por desenhar uma função que desenha as paredes, dadas as suas dimensões, posição e cor, como a seguir mostramos:

```
// Função para criação de uma parede, dando as dimensões, posição e cor.
function createCube( sx, sy, sz, p, cor ) {
var cube = new THREE.Mesh( new THREE.CubeGeometry( sx, sy, sz ),
new THREE.MeshLambertMaterial( { color: cor } ) );
cube.position = p;
paredes.push(cube); // Põe o objecto "cube" (cada parede) num array (previamente criado).
scene.add( cube );
return cube;
}
```

Dando agora um exemplo de chamada desta função:

```
var parede = createCube( 150, 50, 2, new THREE.Vector3(-125, 0, 150), 0x000000 );
```

Na chamada à função, é criado um cubo com a largura de 150, altura de 50 e profundidade de 2, com o seu ponto central situado nas coordenadas (-125,0,150) com a cor, em hexadecimal (neste caso, cor preta).

A necessidade de introduzir cada parede criada num array está relacionada com a implementação equivalente aos sensores de toque, explicados mais adiante. É possível também efectuar rotação (em radianos) a estes objectos, sendo apenas necessário o seguinte código:

```
// Rotação de 90º (PI/2) da parede no eixo dos yy.
parede.rotation.y = (Math.PI/2);
```

Sendo também possível fazer a rotação em torno dos eixos dos xx e zz, mudando a coordenada correspondente no código.

A título de complemento, apresento exemplos de como criar outros objectos, como uma **esfera** ou um **plano**:

Também é possível adicionar, por exemplo, esferas ou planos na cena, existindo, tal como no caso do cubo, que usámos para a construção do modelo da sala deste protótipo, objectos já definidos para a construção rápida destas geometrias.

### 3.6.2 Navegação na cena

A navegação na cena é importante para oferecer ao utilizador uma sensação de navegar livremente na cena, o que enriquece bastante uma visita virtual. Não é de todo conveniente que o utilizador se sinta limitado nesse aspecto, pois afecta a sensação de imersividade e retira a possibilidade do utilizador observar rigorosamente as obras expostas.

Para implementar a navegação na cena optámos por fazê-lo recorrendo ao rato e ao teclado (botões direccionais). Os botões direccionais efectuam translacções na câmara enquanto que com o rato o utilizador pode rodá-da em qualquer sentido, desta forma é dada ao utilizador uma satisfatória liberdade de navegação na cena.

A definição desta navegação é feita da seguinte forma:

```
var controls = new THREE.RollControls( camera );

controls.movementSpeed = 600;

controls.constrainVertical = [ -0.1, 0.1 ];
```

O atributo *movementSpeed* é referente à velocidade com que o movimento é feito a partir das teclas de direcção do teclado. Quanto ao atributo *constrainVertical*, este delimita a rotação da câmara em torno do eixo dos yy (eixo vertical).

A variável controls é actualizada a todo o momento na função render(), dado que se trata de um input interactivo. Esta função render() é chamada na função animate(), sendo esta chamada na *main function* init(). Isto é mostrado a seguir:

```
function animate() {

requestAnimationFrame( animate );

render();

}

function render() {

controls.update();

camera.lookAt( scene.position );

renderer.render( scene, camera );

}
```

O atributo lookAt da câmara mantém o foco da câmara centrado na cena.

### 3.6.3 Sensores de toque

Para implementar Sensores de toque, a ideia foi desencadear uma acção aquando de um clique com o rato. O objectivo é atribuir a determinados objectos da cena a capacidade de alterar algum dos seus atributos (dimensão, cor, etc) com um clique sobre si.

Neste caso, optámos por se mudar a cor de um objecto seleccionado com o rato através de clique, de modo a atribuir a este objecto o "status de seleccionado", mudando a sua cor para vermelho (fig 3.1). Esta tarefa mostra que é possível fazer algo idêntico aos sensores de toque da aplicação Virtual Exhibition Builder. Mostramos um exemplo do efeito de um clique numa das superfícies a figura 3.1:

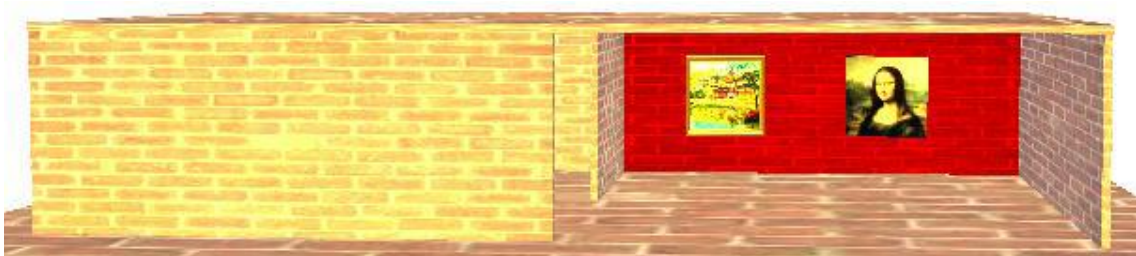



**Figura 3.1 - Efeito do clique numa parede**

Mostramos a seguir a função associada a esta tarefa:

```
function onDocumentMouseDown( event ) {

event.preventDefault();

var vector = new THREE.Vector3( ( event.clientX / window.innerWidth ) * 2 - 1, - (
event.clientY / window.innerHeight ) * 2 + 1, 0.5 );

projector.unprojectVector( vector, camera );

var ray = new THREE.Ray( camera.position, vector.subSelf( camera.position
).normalize() );

var intersects = ray.intersectObjects( paredes );

if ( intersects.length > 0 ) {

// Seleccionado -> Nao seleccionado

if((intersects[ 0 ].object.materials[ 0 ].color.getHex()) == 0xa00000) intersects[ 0
].object.materials[ 0 ].color.setHex( corAnterior );
```

```
// Não Seleccionado -> Seleccionado
else {
corAnterior = intersects[ 0 ].object.materials[ 0 ].color.getHex();
intersects[ 0 ].object.materials[ 0 ].color.setHex( 0xa00000 );
}
}
}
```

Explicando um pouco o que a função faz: na variável “paredes”, do tipo array, encontram-se todos os objectos considerados “parede”. Quando é detectado o evento “clique” num destes objectos, o objecto muda a sua cor para vermelho, cor que representa o estado “seleccionado”, se este objecto for da cor vermelha, isto é, se já foi seleccionado anteriormente, a função devolve a cor original ao objecto, passando este ao estado “não seleccionado”.

O código hexadecimal 0xa00000 é equivalente à cor vermelha.

### 3.6.4 Fontes de luz

Existem alguns tipos de fontes de luz disponibilizadas nesta biblioteca, como a “DirectionalLight”, “AmbientLight” e “SpotLight”. Para a iluminação da cena é usada uma AmbientLight de cor branca e fontes de luz, do tipo “SpotLight”. A decisão pelo tipo de fonte “SpotLight”, tem a ver com a implementação de outra tarefa, a projecção de sombras. Apenas com este tipo de fonte de luz é possível projectar sombras [IntWebGL]. Mostramos, a seguir, a função para a criação de uma fonte de luz:

```
function createLight(color, x, y, z) {
light = new THREE.SpotLight( color );
light.position.set( x, y, z );
scene.add( light );
sphere = new THREE.Mesh( new THREE.SphereGeometry( 5, 5, 5 ), new THREE.MeshLambertMaterial( { color: color } ) );
sphere.position = light.position;
scene.add( sphere );
```

```
}
```

Esta função cria uma fonte de luz do tipo SpotLight, com a cor e posição desejadas. Para uma fácil percepção da posição da luz na cena, representámos uma pequena esfera na posição da luz, com a mesma cor desta.

Fora deste método, na *main function* init(), foi criada uma AmbientLight de cor branca, da seguinte maneira:

```
var sun = new THREE.AmbientLight( 0xffffff );
```

Este tipo de fonte de luz, não necessita de posicionamento, dado que é uma luz ambiente, equivalente, por exemplo à luz do dia que entra pelas janelas de nossas casas.

Existe alguma incoerência no comportamento das fontes de luz em relação aos objectos da cena, como poderá observar-se na figura 3.2:

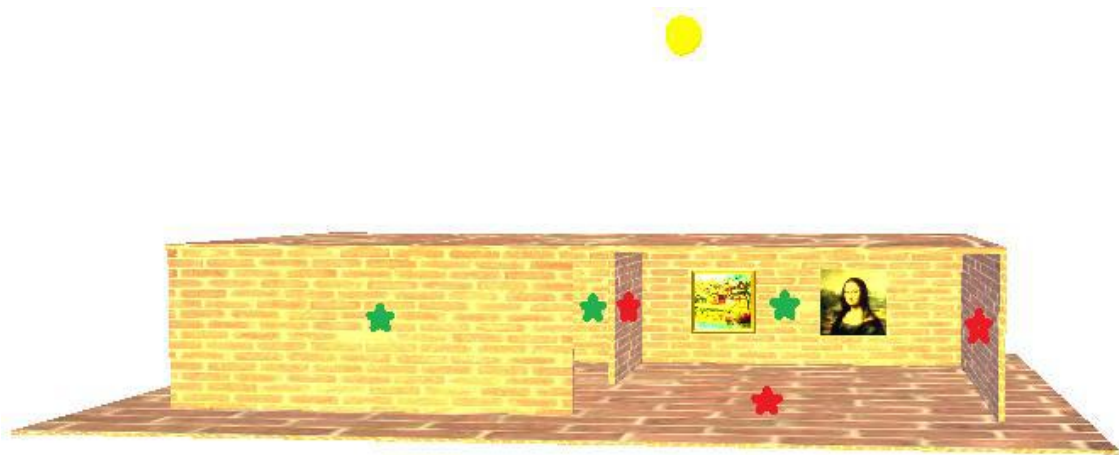



**Figura 3.2 - Inconsistências na iluminação da cena**

Na figura, os objectos assinalados a vermelho, deveriam ter o mesmo aspecto que os objectos assinalados a verde, dada a posição da fonte de luz, representada pela esfera amarela no topo da imagem. Apenas a face dos paralelepípedos, que representam as paredes, virada para a fonte de luz está iluminada. Note que a fonte de luz está de frente para as obras, estando mais alta que o modelo da sala.

O screenshot da figura não tem a projecção de sombras activada. Veremos a projecção de sombras na sub-secção seguinte.

### 3.6.5 Projecção de sombras

A projecção de sombras é uma das tarefas testadas mais importantes deste protótipo, dado que a projecção de sombras foi um dos maiores motivos para a experimentação da tecnologia WebGL. Parte importante da decisão acerca do uso desta tecnologia estará relacionada com esta componente.

Como foi mencionado na descrição da tarefa anterior, o que permite a projecção de sombras são as fontes de luz do tipo “SpotLight” [IntWebGL].

Para a representação das projecções de sombra, é necessário “informar” as fontes de luz que estas “originam” sombras na cena. Com os objectos da cena, é necessário fazer algo parecido, isto é, “informá-los” que podem “receber” e “projectar” sombras. Isto faz-se após a criação de cada fonte de luz “SpotLight” e de cada objecto da cena (excepto a câmara).

Demonstrando:

```
scene.add( light );
light.castShadow = true;
scene.add( parede );
cube.castShadow = true;
cube.receiveShadow = true;
```

Finalmente, é necessário adicionar ao renderer a possibilidade de representar as sombras. Isso é feito da seguinte forma:

```
renderer.shadowMapEnabled = true;
renderer.shadowMapSoft = true;
```

Porém, existem algumas incoerências no que diz respeito ao comportamento das sombras nos objectos, como poderá visualizar na figura 3.3:

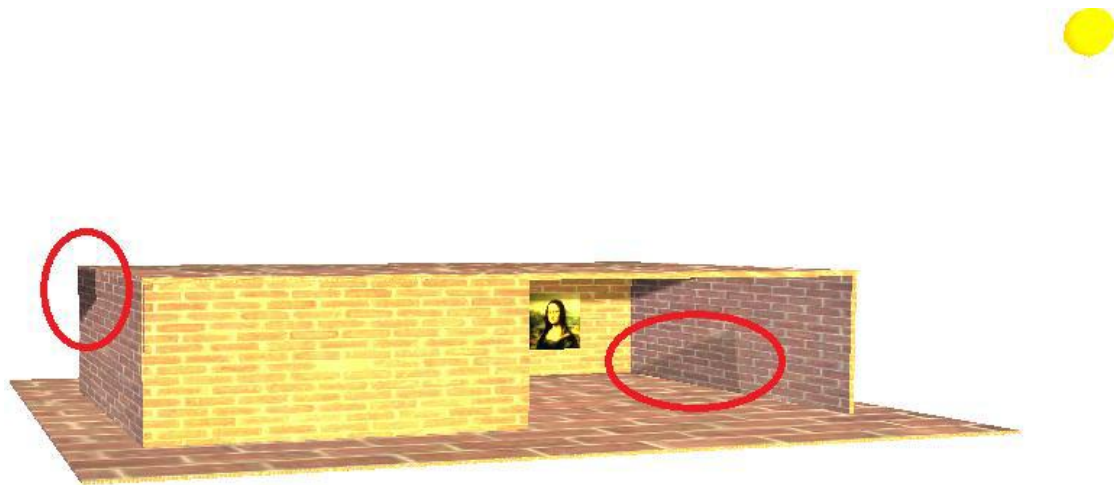



**Figura 3.3 - Inconsistências nas sombras relativamente à fonte de luz**

As sombras assinaladas a vermelho não estão em conformidade com a geometria da sala e com a posição da fonte de luz.

### 3.6.6 Interface de interacção com objectos da cena

Para a implementação da Interface de interacção com objectos da cena, foi usada uma outra biblioteca, denominada Dat.GUI.

Esta tarefa foi adicionada devido à necessidade de ajustar posicionamentos das obras de arte, isto é, por exemplo, ajustar a posição de um quadro numa parede.

Para isso, basta criar um objecto do tipo DAT.GUI, e associá-lo a um objecto, da seguinte forma:

```
var gui = new DAT.GUI();
gui.add(quadro.position, 'x').min(-100).max(100).step(1);
gui.add(quadro.position, 'y', -100, 100, 1);
gui.add(quadro.position, 'z', 100, 800, 1);
```

Este de código cria um menu na cena que permite a mudança de posição de um quadro em cada eixo, de forma interactiva. De seguida, mostramos um exemplo de uso nas figuras 3.5 e 3.6:

**Figura 3.4 – Cena original.**

**Figura 3.5 – Cena alterada.**

Como é visível nas figuras 3.4 e 3.5, os quadros têm as posições trocadas na parede, para isso apenas foram alteradas as suas posições em x (1ª barra azul). O quadro inicialmente à esquerda (figura 3.4), cuja interface é a da esquerda em ambas as figuras, viu a sua posição no eixo dos xx aumentada, isto é, foi deslocada para a direita, tendo-se feito o contrário para o outro quadro, sendo o resultado dessas alterações visíveis na figura 3.5. As posições de ambos os quadros em y e z permanecem alteradas.

Não sendo muito visível, nas figuras esclarecemos que na interface, cada barra corresponde a um eixo principal: x,y,z de cima para baixo.

### 3.6.7 Texturas

A funcionalidade de introduzir texturas é importante para este trabalho. Estas permitem aumentar o grau de realismo dos modelos tridimensionais, tal como atribuir uma textura às paredes do modelo tridimensional ou a introdução de quadros na sala.

A solução encontrada foi a criação de um "Material" associado a uma imagem que servirá de textura para o objecto ao qual o material está ligado. Para usar a imagem basta colocar o endereço do objecto no pedaço de código a seguir.

```
var materialML = new THREE.MeshLambertMaterial({

map: THREE.ImageUtils.loadTexture("Endereço da imagem")

});
```

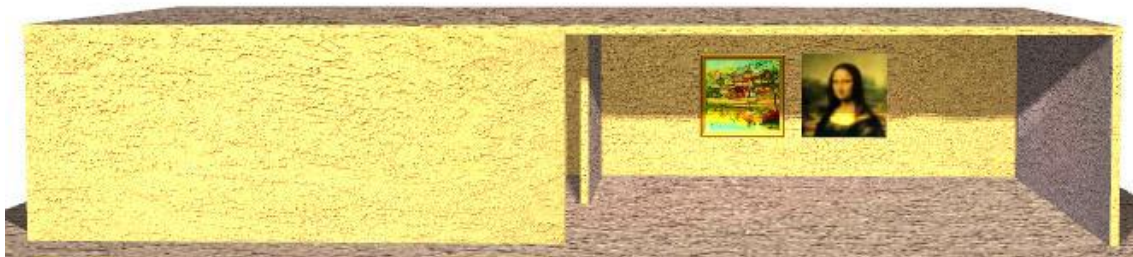



**Figura 3.6 - Cena com texturas.**

A figura 3.6 tem associadas texturas em todos os objectos, isto é, em todas as superfícies (paredes) e nos objectos que representam as obras.

## 3.7 Discussão

Nesta secção apresentamos as conclusões do protótipo feito usando a tecnologia WebGL.

Quanto à ligação a uma base de dados, existe o problema do WebGL correr num Web browser, o que impossibilita a máquina do browser de comunicar com uma base de dados directamente, por razões de segurança. A melhor maneira de obter informação de uma base de dados nestas condições é criar uma aplicação server-side usando uma linguagem para esse fim, por exemplo, o PHP [blogWebGL]. Esta aplicação "server-side" fará a ponte entre o lado do cliente (JavaScript) e a Base de Dados em si, isto é,

receberá as queries enviadas do cliente, e encaminhará os resultados das consultas de novo para ele.

Como foi mostrado na secção 3.6.5, a projecção de sombras não tem o comportamento esperado, sendo inconsistente com as geometrias e posições de fontes de luz testadas. Juntando a este facto o problema da falta de experiência no uso da tecnologia WebGL e JavaScript, o desenvolvimento de uma nova aplicação foi abandonado, optando-se por estender a aplicação já construída, a Virtual Exhibition Builder.

No capítulo seguinte, iremos descrever as extensões efectuadas na aplicação.

# Capítulo 4 Trabalho Desenvolvido – X3D

Neste capítulo descrevemos as funcionalidades acrescentadas à aplicação Virtual Exhibition Builder [Gomes11], nomeadamente: uma componente para a criação e actualização interactiva de uma base de dados de obras de arte; alteração do modo de salvaguarda de exposições; visualização dos valores das distâncias dos objectos de arte a pontos de referência; a adição de vitrines no espaço da exposição; a construção de representações de obras de arte 3D de forma simplificada sem recurso a modelação tridimensional e a criação de um ficheiro de configurações que permite o ajuste de alguns parâmetros por omissão da aplicação de forma fácil para o utilizador.

Algumas das funcionalidades que iremos descrever a seguir, envolveram manipulação de elementos do grafo da cena, tendo sido o [Sledz07], uma ajuda preciosa para perceber o uso do SAI (Scene Access Interface).

A programação foi feito na linguagem Java. A aplicação Virtual Exhibition Builder foi estendida recorrendo ao Java SE JDK (Java Development Kit) versão 7. O IDE (Integrated Development Environment) foi o Eclipse Indigo [Eclipse]. As bases de dados foram concretizadas com SQLite, uma biblioteca que implementa uma base de dados SQL embutida. Esta ferramenta permite aceder a bases de dados sem executar um processo SGBD separado. [sqliteAbout].

## 4.1 Componente de criação e actualização de bases de dados

A aplicação existente estava ligada a uma base de dados específica. Para permitir a construção de exposições associadas a diferentes colecções de arte, foi necessários criar uma forma de associar diferentes bases de dados ao mesmo espaço físico de um museu.

Para que a concretização desta tarefa fosse possível, tivemos de alterar a forma como a conexão à base de dados era feita. Na aplicação anterior o nome da base de dados estava definido no código, com o nome de “sqlitedb”.

Para a aplicação poder aceitar diferentes base de dados, teve de se ter em consideração um mecanismo de conexão que permita associar identificadores diferentes. Antes o método que ligava a base de dados não tinha qualquer argumento, ou seja, não recebia qualquer informação, dado que ligava sempre à base de dados “sqlitedb”. O que fizemos foi transformar o método de forma a aceitar um nome de uma base de dados e ligar-se a ela, permitindo desta forma que um mesmo espaço físico possa ter associadas vários repositórios.

## Aplicação de criação e actualização de bases de dados

Criou-se uma aplicação interactiva para a criação e actualização de bases de obras de arte.

A actualização da base de dados estava a cargo de uma aplicação pouco amigável para o utilizador comum, o SQLite Studio.

Para resolver este problema, decidimos implementar uma solução que oferece uma interface mais amiga do utilizador para criação e actualização de bases de dados.

Foi debatido como seria mais correcto implementar esta funcionalidade como uma opção de menu da aplicação Virtual Exhibition Builder ou como uma aplicação separada. Decidimos criar uma aplicação que pode ser executada autónomamente mas que também pode ser invocada através de uma opção na aplicação Virtual Exhibition Builder.

A base desta decisão esteve na ideia de separar claramente duas fases de concepção de uma exposição: a construção da base de dados do reportório e a montagem da exposição.

No entanto, entendemos ser conveniente oferecer ao utilizador a possibilidade de fazer ajustamentos na base de dados na fase de construção da exposição, isto é, já durante a montagem desta, caso durante este processo o utilizador se depare com qualquer erro não detectado aquando da construção e preenchimento da base de dados. Esta possibilidade é oferecida através de uma opção no menu principal (Fig. 4.1).

A ferramenta relativa à criação e actualização de bases de dados foi desenvolvida com base no Java Swing.

## Esquema relacional simplificado da aplicação

Antes de procedermos à demonstração das interfaces desenvolvidas, mostramos de seguida a estrutura da base de dados usada na aplicação:

**ArtType**(type_id, type_name)

**Artwork2D**(art_id, title, author, year, type, height, width, image_path, description, observations)

**Artwork3D**(art_id, author, year, model_path, title, bbox_upper, bbox_lower, image_path, height, width, length, description, observations)

**Authors**(author_id, author_name, author_nationality)

Os campos assinalados a verde são os atributos adicionados à estrutura da base de dados no nosso projecto, relativamente à aplicação anterior.

Em relação ao esquema relacional de base de dados anteriormente definido, as alterações introduzidas correspondem à adição dos campos *description* e *observations* às tabelas Artwork2D e Artwork3D. Estes campos são do tipo texto e irão conter informação complementar sobre as obras 2D e 3D.

O campo *description* irá conter informação adicional sobre uma obra, ou seja, é um campo criado para proporcionar aos utilizadores a possibilidade de observar e recolher informações sobre a obra que estão a ver numa visita virtual a uma exposição criada nesta aplicação.

O campo *observations* é destinado a quem constrói a exposição, isto é, não irá conter infromação do interesse do utilizador que "visita" a exposição, mas sim de quem a irá conceber. A informação contida neste campo poderá ser relativa à natureza das próprias obras, por exemplo, informará a quem está a conceber a exposição, que a exposição de determinado objecto será com uma vitrine.

## Interface com o utilizador

Passamos agora a mostrar as interfaces desenvolvidas, bem como, a forma como o utilizador interage com elas.

Em primeiro lugar, aquando da execução da aplicação de criação e actualização de bases de dados, uma janela surgirá, com duas opções: *Create Database* e *Open Database*.

A opção *Create Database*, tem como finalidade, como o próprio nome indica, criar uma base de dados. O utilizador terá a liberdade de criar e guardar o ficheiro relativo à base de dados onde quiser, através de um *File Chooser*, tendo como pré-definição a pasta "databases" na pasta raíz do projecto. O utilizador define um nome nesse *File Chooser* e a aplicação criará um ficheiro no local indicado pelo utilizador com a extensão ".db3". Após criar o ficheiro da base de dados, é aplicado o *DDL (Data Definition Language)* sobre esse ficheiro, DDL esse que corresponde ao esquema relacional da base de dados descrito atrás. Após este processo, uma outra janela se abrirá com 4 opções, usando o ficheiro (base de dados) criado: *Insert 2D Data, Insert 3D Data, Consult/Edit 2D Data* e *Consult/Edit 3D Data*.

As opções de adicionar obras 2D e 3D estão separadas dado que os formulários são diferentes, tal como os campos dessas tabelas. É de destacar que, no formulário de inserção de obras 3D, existem duas variantes, devido aos tipos de representação dessas obras: detalhado e não detalhado. Há a hipótese de uma obra ter uma das duas representações ou mesmo ambas. Os modelos detalhados são elaborados em X3D, já os não detalhados são modelos contruídos posteriormente em função das dimensões e imagens fornecidas, como explicaremos mais detalhadamente na secção 4.5.

Esta separação entre as obras 2D e 3D sucede também com as opções de consulta/actualização das obras, pelas mesmas razões da separação para a inserção, ou seja, os campos são diferentes. Destacamos ainda o facto da informação para consulta/actualização estar disposta em forma de tabela, sendo que as colunas desta representam os campos das tabelas (Artwork 2D e 3D) e as linhas representam as ocorrências da mesma.

As opções *Insert* servem exclusivamente para inserir dados, já os campos Consult/Edit podem ser utilizados para consulta, actualização e eliminação de ocorrências.

A opção *Open Database*, permite abrir uma base de dados previamente construída. Uma vez escolhido o ficheiro através de um *File Chooser*, tal como na opção descrita antes desta, é aberta uma janela com as opções de criar, consultar e actualizar dados relativos a obras 2D e 3D.

Como já foi referido, a actualização de bases de dados de obras de arte também pode ser feita através de uma opção de menu da aplicação Virtual Exhibition Builder. Neste caso, a actualização é feita sobre a base de dados associada à exposição. Essa opção encontra-se no menu principal, na opção databases, como mostra a figura 4.1:

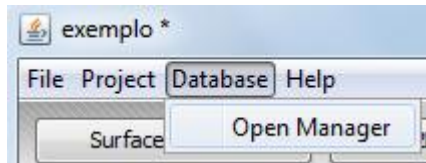



**Figura 4.1 - MenuItem base de dados.**

Existem 2 ficheiros executáveis na nossa aplicação, o primeiro, relativo à ferramenta de construção de exposições, e o segundo, relativo à criação e manutenção de bases de dados.

Ao executar a 2ª, abrirá uma janela que permitirá ao utilizador criar ou aceder a uma base de dados existente. Nas duas opções surgirá um *File Chooser*, que permitirá ao utilizador criar ou abrir o ficheiro em qualquer localização. A interface corresponde à figura 4.2:

**Figura 4.2 - Interface de criação e edição de bases de dados.**

Os *File Choosers* abrirão, por omissão, na pasta "databases", localizada na pasta raíz da aplicação.

Ao criar ou aceder a uma base de dados, esta janela é fechada e abrirá outra, com 4 opções: Add 2D Arts, Add 3D Arts, Edit 2D Arts e Edit 3D Arts, como representado a seguir pela figura 4.3:

**Figura 4.3 - Janela de manutenção de bases de dados.**

A nome da base de dados seleccionada no momento é mostrada como acima, a verde, para que o utilizador tenha a certeza de que está a manipular a base de dados correcta.

As opções *Add* oferecerão uma interface que o utilizador poderá usar para inserir obras 2D (Quadros, Tapeçarias) e obras 3D.

Em ambos os casos, surgem formulários para o utilizador preencher, segundo os campos da tabela correspondente.

Apresentamos de seguida, as figuras 4.4 e 4.6, que representam as opções de inserção de dados de obras (2D e 3D):

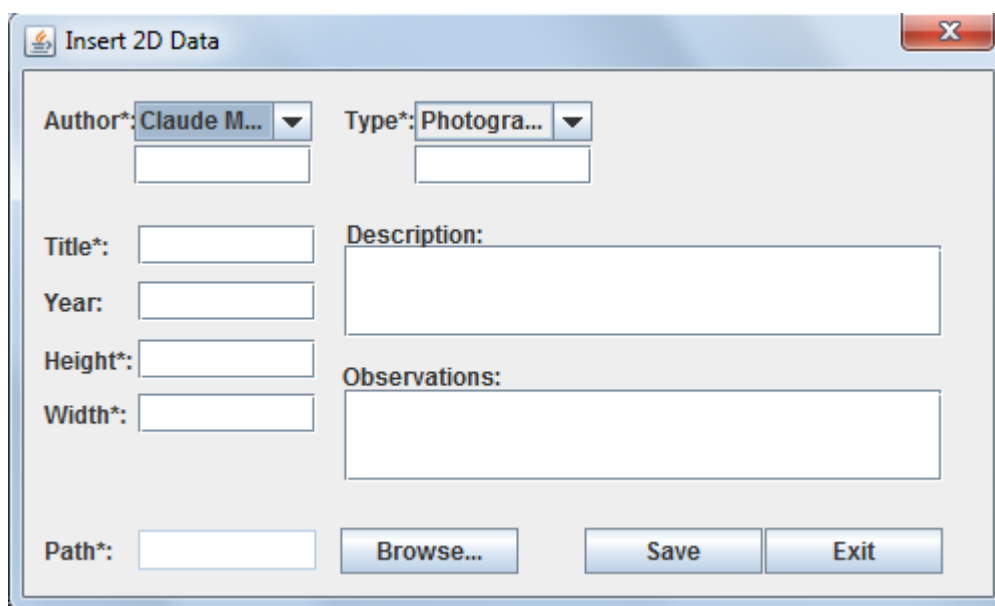



Figura 4.4 - Janela de inserção de dados das obras 2D.

Como se pode ver na figura 4.4, foi feita uma interface bastante simples, pedindo ao utilizador os dados da obra e a imagem (sua localização no disco), que pode procurar através do botão *Browse*.

Para ajudar na inserção de obras com um autor ou tipo de obra já presentes na base de dados, são diponibilizadas essas informações em *Combo Boxes*, com os dados existentes referentes a esses campos, como mostramos a seguir na figura 4.5:

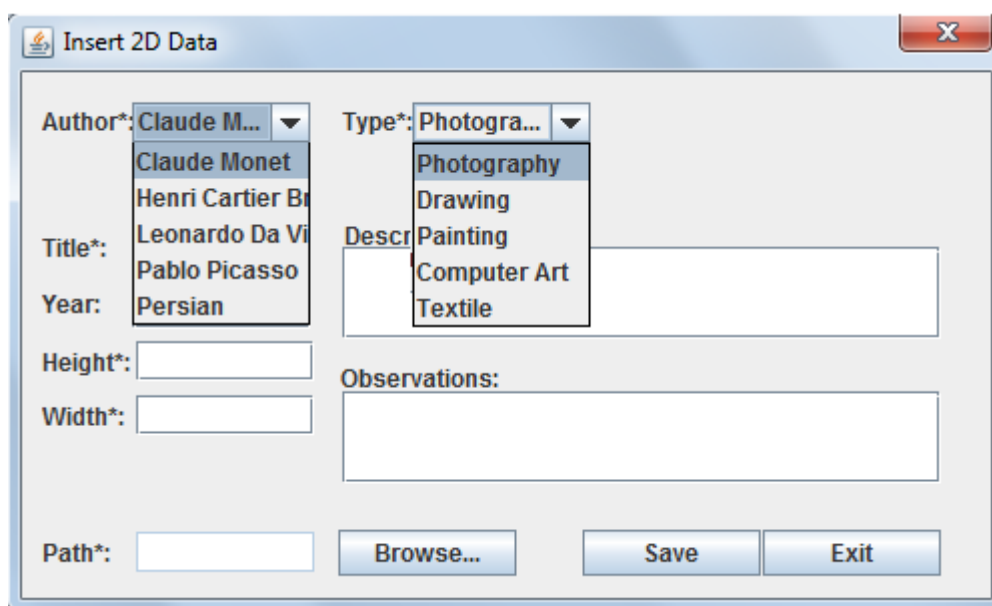



Figura 4.5 - Exemplo de preenchimento prévio de *Combo boxes.*

Ao seleccionar uma das opções em cada *combo box*, a caixa de texto em baixo ficará preenchida com o nome seleccionado. Caso não o queira fazer, o utilizador terá de introduzir manualmente o nome do novo autor/tipo de obra nessa mesma caixa de texto.

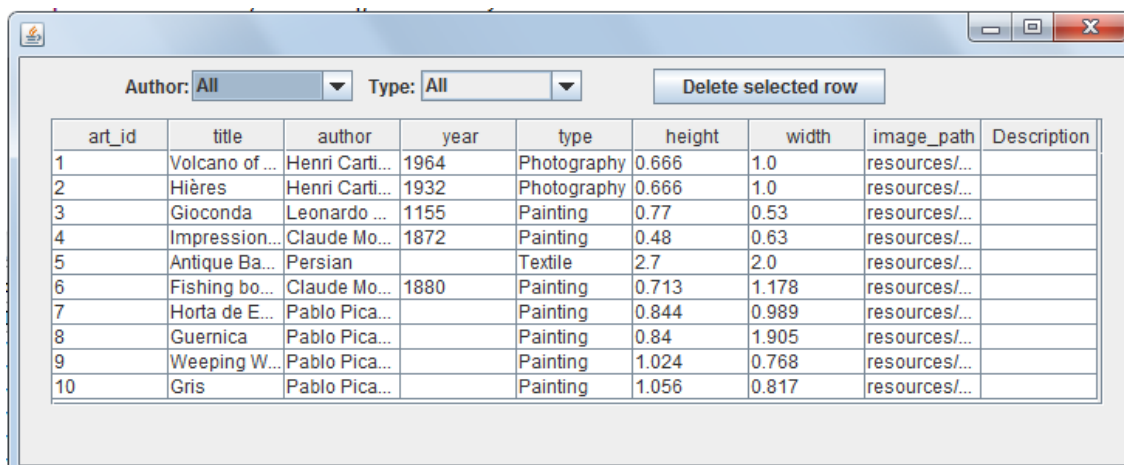
Mostramos agora a interface de inserção de obras 3D, na figura 4.6:

**Figura 4.6 - Janela de inserção de dados de obras 3D.**

O que difere desta janela para a outra são os campos de cada tabela, que são diferentes da tabela de obras 2D. É importante referir que existem 2 tipos de obras 3D, baseadas em modelos X3D previamente feitos (*high detail*) e baseados em imagens (*low detail*). Existem casos em que uma obra pode ter ambos os tipos de representação.

Sendo assim é essencial que o utilizador forneça informação para a obra ter, pelo menos, um tipo de representação, ou seja, no caso de não serem inseridas quaisquer imagens às faces do cubo, tem de ser inserido um modelo, e caso não seja introduzido nenhum modelo, pelo menos uma imagem do objecto deverá ser fornecida. No caso de ser fornecida apenas uma imagem, essa imagem será colocada em todas as faces da

*bounding box* do objecto, caso contrário, as imagens fornecidas serão colocadas nas faces escolhidas pelo utilizador.

Acrescentamos que as obras 3D baseadas em imagens terão apenas a forma de um paralelepípedo, isto é, serão aceites, no máximo, 6 imagens, equivalente ao número de faces do paralelepípedo.

De referir que, os campos marcados com *, são os campos de preenchimento obrigatório.

Será lançada uma mensagem de erro se algum dos campos assinalados com * não estiver preencido e se o tipo de dados colocado em cada campo não corresponder ao correcto, por exempo, colocar letras num campo numérico. Esta regra aplica-se também à interface de inserção de obras 2D.

As opções *Edit* permitem ao utilizador consultar e actualizar campos das tabelas de obras 2D e 3D, tal como eliminar ocorrências da base de dados, isto é, eliminar obras. A informação é disponibilizada através de tabelas, que serão preenchidas conforme as informações que o utilizador fornecerá aos filtros configurados em cada modo de actualização. Na actualização de obras 2D existem 2 filtros: Autor e Tipo de obra, e na actualização de obras 3D existe um filtro, apenas por autor, dado que, na base de dados, não estão associados tipos de obra nas obras 3D (assume-se que são apenas esculturas).

Existe uma separação clara entre 2D e 3D dado que a estrutura das tabelas é diferente, pelo que os campos e informações apresentados em cada interface também o serão.

As figuras 4.7 e 4.8 apresentam a interface das 2 opções de actualização de bases de dados:

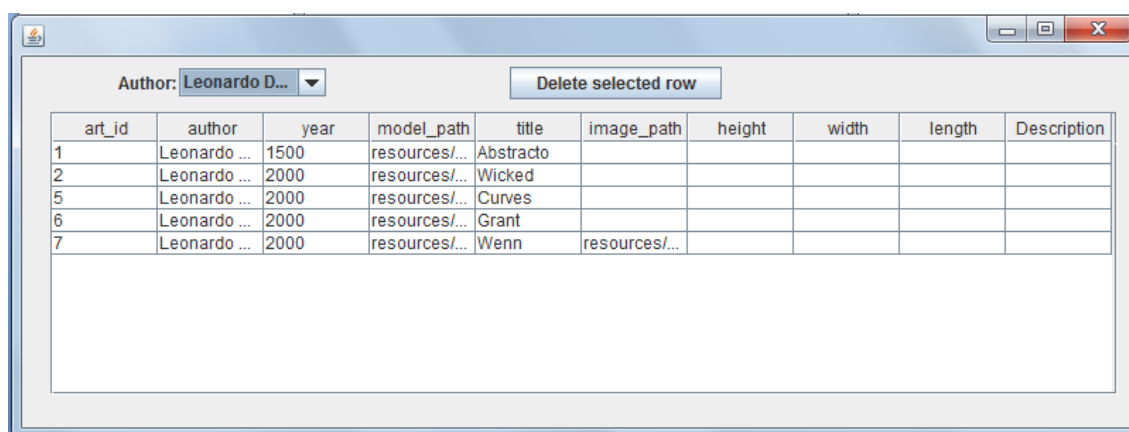
Author: All Type: All Delete selected row

| art_id | title | author | year | type | height | width | image_path | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | Volcano of ... | Henri Carti... | 1964 | Photography | 0.666 | 1.0 | resources/... | |
| 2 | Hières | Henri Carti... | 1932 | Photography | 0.666 | 1.0 | resources/... | |
| 3 | Gioconda | Leonardo ... | 1155 | Painting | 0.77 | 0.53 | resources/... | |
| 4 | Impression... | Claude Mo... | 1872 | Painting | 0.48 | 0.63 | resources/... | |
| 5 | Antique Ba... | Persian | | Textile | 2.7 | 2.0 | resources/... | |
| 6 | Fishing bo... | Claude Mo... | 1880 | Painting | 0.713 | 1.178 | resources/... | |
| 7 | Horta de E... | Pablo Pica... | | Painting | 0.844 | 0.989 | resources/... | |
| 8 | Guernica | Pablo Pica... | | Painting | 0.84 | 1.905 | resources/... | |
| 9 | Weeping W... | Pablo Pica... | | Painting | 1.024 | 0.768 | resources/... | |
| 10 | Gris | Pablo Pica... | | Painting | 1.056 | 0.817 | resources/... | |

**Figura 4.7 - Interface de consulta/edição de base de dados (Obras 2D).**

Como acima foi referido, existem 2 filtros, que são representados por 2 *combo boxes* preenchidas com os autores e tipos de obra disponíveis na base de dados, como é visível na imagem acima. Neste exemplo, são apresentadas todas as obras 2D presentes na base de dados, dado não é especificado nenhum autor nem tipo de obra.

Existe também a possibilidade de eliminar ocorrêcias, bastando para isso seleccionar quaquer uma das linhas e carregar no botão “Delete selected row”.

| art_id | author | year | model_path | title | image_path | height | width | length | Description |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Leonardo ... | 1500 | resources/... | Abstracto | | | | | |
| 2 | Leonardo ... | 2000 | resources/... | Wicked | | | | | |
| 5 | Leonardo ... | 2000 | resources/... | Curves | | | | | |
| 6 | Leonardo ... | 2000 | resources/... | Grant | | | | | |
| 7 | Leonardo ... | 2000 | resources/... | Wenn | resources/... | | | | |

**Figura 4.8 - Interface de consulta/edição de base de dados (Obras 3D).**

Neste caso, representado pela figura 4.8, o filtro está definido para mostrar todas as obras 3D do autor “Leonardo Da Vinci”.

Todos os campos que não são chave são editáveis, excepto os campos preenchidos automaticamente pela aplicação, como as informações referentes às *bounding boxes* das obras, informações essas que não é conveniente editar e nem são do interesse do utilizador, sendo esses campos ocultados nas opções de criação e actualização de bases de dados. Para editar um campo, basta efectuar um duplo clique na célula correspondente e alterar o seu conteúdo. Caso o utilizador insira dados diferentes do tipo de dados desse campo, a alteração não será efectuada na base de dados e será lançada uma mensagem de erro.

Ora, toda esta ideia de facilitar a criação e actualização de bases de dados tem a ver com a possibilidade de o utilizador poder associar uma base de dados feita por si a uma exposição. Devido a isso, na criação de uma exposição, é pedido ao utilizador que dê um nome à exposição e que associe a ela um modelo tridimensional (em X3D) do espaço físico e uma base de dados, como poderá observar na figura 4.9:

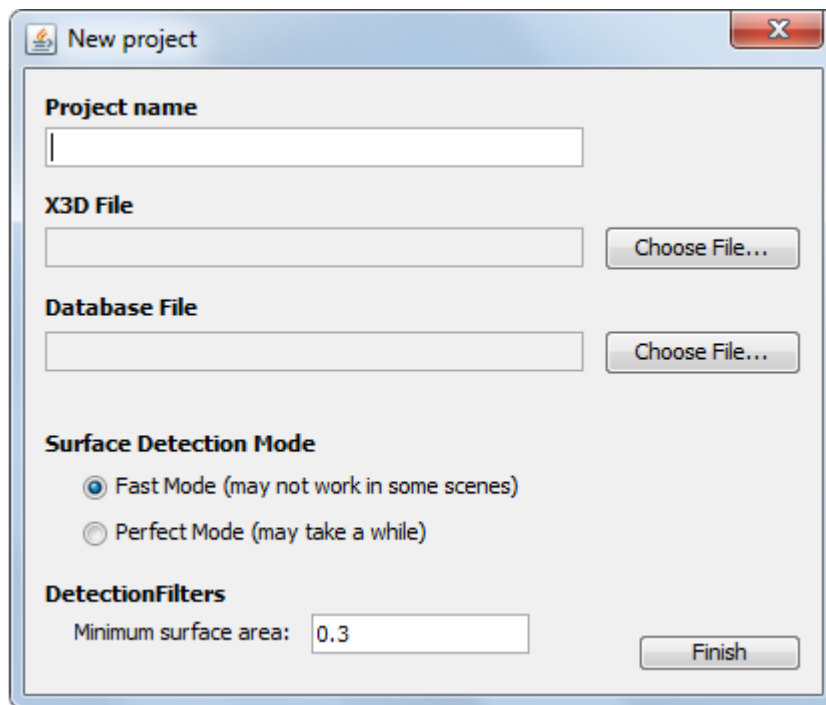



**Figura 4.9 - Janela de criação de nova exposição.**

## 4.2 Salvaguarda de Exposições

Agora vamos falar das alterações feitas na salvaguarda de exposições. Esta tarefa tem uma ligação directa com a criação de mais do que uma base de dados de obras de arte. É possível ter vários repositórios de dados que podem ser associados a qualquer exposição.

Temos 3 opções para o utilizador guardar o trabalho feito: o *Save*, *Save As* e *Save Only Exhibition*.

As duas primeiras opções já estavam implementadas na aplicação Virtual Exhibition Builder. Fizemos uma alteração no *Save as* e criámos o *Save only exhib*.

O *Save as* abria um *FileChooser* que permitia o utilizador gravar um ficheiro de estado onde quisesse.

A alteração feita no *Save As* tem a ver com o facto de agora existir noção de atribuir qualquer repositório de dados a uma exposição, ou seja, a cada ficheiro de estado terá associado o seu ficheiro da base de dados. Posto isto, o nosso *Save As* guarda 2 ficheiros. O ficheiro de estado e o ficheiro da base de dados. O utilizador é convidado a

escolher o local onde quer que a exposição seja guardada, escolhe um nome e nesse local a aplicação criará uma pasta com o nome indicado com os ficheiros de estado e base de dados com o mesmo nome, mas obviamente com diferentes extensões. Exemplificando, o utilizador, com o *File Chooser*, navega até ao *Desktop* e escolhe como nome da exposição, "teste". Depois disto, a aplicação cria uma pasta chamada "teste" e lá dentro guardará 2 ficheiros, o "teste.vgp" e o "teste.db3", respectivamente, o ficheiro de estado da exposição e o ficheiro da sua base de dados.

A opção *Sabe only exhib* foi criada para o utilizador não ter sempre de gravar 2 ficheiros aquando da edição de um ficheiro de estado, ou seja, esta opção apenas é disponibilizada quando a exposição já foi guardada pelo menos 1 vez ou se a exposição foi aberta na aplicação utilizando a opção *open*. Assim, o utilizador pode guardar apenas o ficheiro de exposição ou então criar novas exposições (novas arrumações das obras, por exemplo) usando a mesma base de dados.

Com esta modificação, podem então surgir casos em que uma pasta de uma exposição poderá ter 1 ficheiro de bases de dados e vários ficheiros de estado, não havendo necessiade de haver vários ficheiros iguais da base de dados, o que poderia gerar uma ocupação exagerada de espaço no disco, sendo este problema especialmente grave no caso de existirem bases de dados muito grandes. Apresentamos estas opções, presentes no Menu File, como mostramos na figura 4.10:

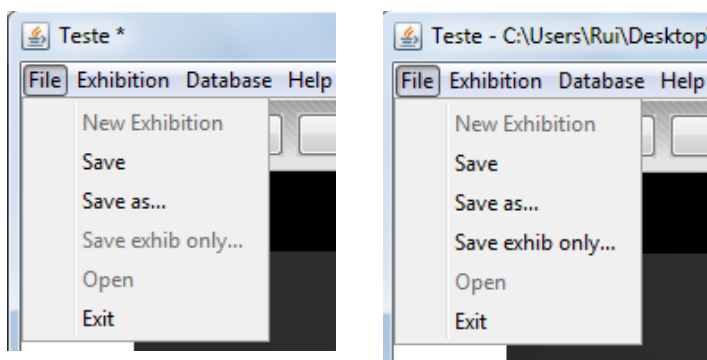



**Figura 4.10 - MenuItem File.**

A imagem da esquerda corresponde a uma exposição que nunca foi salva, devido a isso, a opção *Save exhib only* está desactivada. A imagem da direita corresponde a uma exposição anteriormente guardada, aí a opção *Save exhib only* já está disponível.

Devido à implementação das classes *Art* e *Wall*, que por sua vez, são essenciais para uma correcta visualização dos valores das distâncias, tivemos necessidade de guardar todas as instâncias destas classes para futura edição, juntando estas às informações dos

estados de cada módulo, para que as distâncias possam ser mostradas aquando de uma eventual re-edição de uma exposição.

## 4.3 Visualização de valores de distâncias

Descrevemos nesta secção a funcionalidade, que é um dos principais *upgrades* da aplicação e que foi aconselhada pela equipa de técnicos do Museu da Cidade da Câmara Municipal de Lisboa, que consiste em visualizar, para cada obra de arte, a distância que a separa do limite da superfície onde foi colocada.

Para calcular as distâncias das obras em cada parede, em primeiro lugar é necessário conhecer a posição da mesma. Para isso, começamos por obter as coordenadas dos quatro cantos da parede. Na realidade apenas precisamos da posição de dois, dado que consideramos que apenas são necessárias duas referências para termos informação suficiente para uma colocação correcta da obra numa exposição real.

Começamos por explicar o processo relativo às paredes em si.

Existe um sistema de coordenadas XYZ associado a cada cena, ou seja, a localização dos objectos é descrita sobre a localização nestes eixos.

Optámos por escolher o chão e a extremidade direita de cada parede como referências para a marcação das distâncias, isto significa que iremos medir a distância de cada obra relativamente ao chão e à extremidade direita da parede em que está colocada.

Para determinar a posição em que está o chão, analisàmos as coordenadas dos 4 cantos da parede e obtivemos o valor de y do ponto mais baixo. Para a determinação da extremidade direita de cada parede é necessário ter em conta a sua posição relativamente à cena, isto é, é necessário conhecer a orientação do seu vector normal. Isto porque, para determinar a posição da extremidade direita da parede é necessário saber em que coordenada a obter, isto é, o eixo segundo o qual podemos tirar o seu valor. Consideramos o exemplo da figura 4.11:

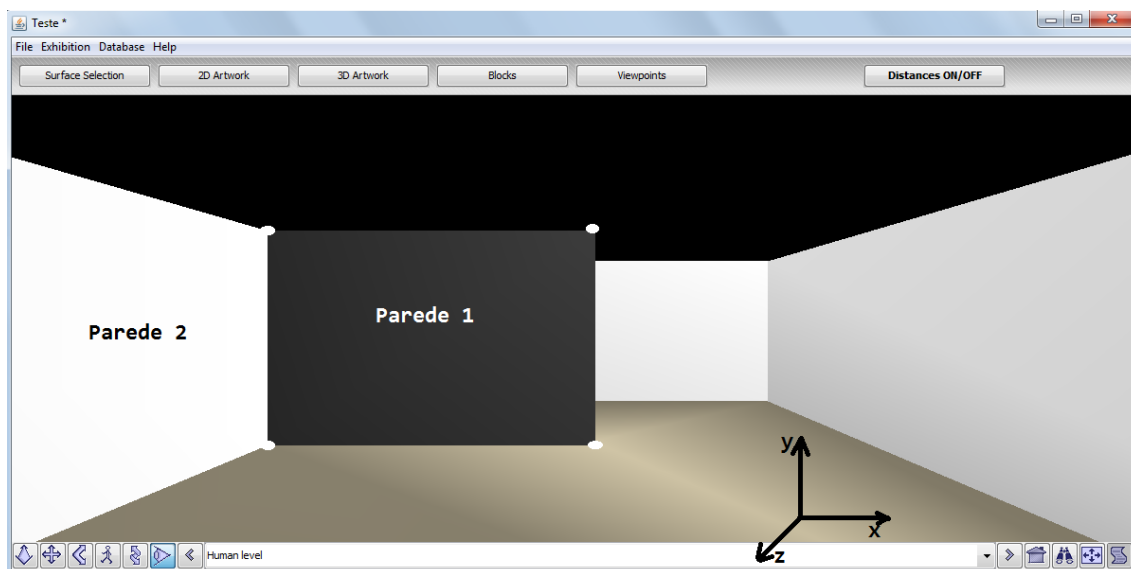



**Figura 4.11 - Exemplo de uma cena sem obras.**

O eixo de coordenadas, os nomes das paredes e as esferas a representar os cantos não são originais da cena, são apenas usados para ajudar no racioncínio.

No caso da parede 1, a mais escura, a extremidade direita da parede, segundo o eixo desenhado na cena, é equivalente ao maior valor x dos 4 cantos da parede.

No caso da parede 2, se o utilizador estiver de frente para ela, a extremidade direita da parede corresponde ao menor valor z dos 4 cantos da parede, e assim por diante.

Relativamente ao chão, este é calculado da mesma maneira nas 2 paredes, ou seja, é obtido o menor valor y dos 4 cantos das paredes.

Este processo, isto é, o cálculo das posições do chão e da extremidade direita de uma parede, é feito para cada parede seleccionada pelo utilizador, seja manual ou automaticamente.

Foi criada uma classe para representar cada parede, a classe *Wall*. Esta classe regista algumas informações de cada parede. Esta informação é essencialmente usada para facilitar o cálculo das distâncias.

A informação registada é a seguinte:

Id – Corresponde ao hashcode da *Surface* que representa a parede.

Normal – Vector que representa a normal da superfície.

MidPt – Ponto central da parede.

Floor – Posição em que está o chão.

Right – Posição em que está a extremidade direita da parede.

Arts – Obras que estão na parede.

Para o campo id é usado o *hashcode* da *Surface* que representa a parede, isto porque esse valor é único e nunca se altera, o que o torna ideal para representar a unicidade deste objecto.

Ora, são criadas tantas instâncias desta classe quantas paredes forem seleccionadas para colocar obras.

Concluído este processo, procedemos à colocação das obras nas paredes.

Para auxiliar no cálculo das distâncias foi criada outra classe, denominada *Art*, que representa cada obra colocada na exposição e que contém informação relevante para esse cálculo.

A informação armazenada em cada instância da classe *Art* é a seguinte:

| Art – Título da obra. |
|---|
| Heigth – Altura da obra (em metros). |
| Width – Largura da obra (em metros). |
| Points – Coordenadas dos 4 cantos das obras. |

Cada vez que as obras são deslocadas, as coordenadas dos seus vértices são actualizadas, assim, as marcações de distâncias estão sempre em conformidade com a posição corrente das obras.

Como já foi referido atrás, são calculadas as distâncias relativamente ao chão e à extremidade direita da parede. Para cada uma das referências existe um algoritmo.

Foi criada uma nova classe, chamada *CalcDistances*, que executa todos os cálculos, desde a determinação das distâncias até ao desenho dessa informação na cena.

Os dois métodos chave desta classe são o *GetFloorDistances* e o *GetRightDistances*.

Este facto deixa claro que o cálculo e desenho das distâncias estão separados em duas partes: o desenho das distâncias das obras relativamente ao chão e o desenho das distâncias relativamente à extremidade direita da parede. Isto deve-se ao facto de, no caso das distâncias para a extremidade direita das paredes, os vectores normais terem de entrar no cálculo. Este factor tem de se ter em conta devido ao facto de a extremidade direita de uma parede depender da posição da mesma na cena, tal como já foi explicado atrás.

Mas não é só este facto que contribui para a separação da tarefa em dois. A forma de desenhar a informação na cena, isto é, o correcto posicionamento dos traços e números, também exige esta separação.

Voltemos agora um pouco atrás, para procedermos à explicação do que foi feito para realizar esta tarefa, o cálculo e desenho das distâncias na cena.

A classe *CalcDistances* recebe 4 argumentos:

| |
|---|
| Wall – Identificador da parede. |
| Normal – O vector normal da parede. |
| Floor – Altura a que o chão está (em metros). |
| Right – Posição em que a extremidade direita da parede está (em metros). |

O argumento wall é o identificador da parede sobre a qual vamos medir as distâncias, o argumento normal é o vector normal da parede, este tem especial utilidade para o cálculo das distâncias na extremidade direita das paredes. Os argumentos floor e right são usados para o cálculo das distâncias.

Esta funcionalidade, a visualização da informação das distâncias, é executada através de um botão colocado no painel de módulos da aplicação, com o nome de *Distances On/Off*. A colocação do botão nesse lugar facilita o acesso a esta funcionalidade em qualquer altura do processo de montagem de uma exposição virtual nesta aplicação, pois é o único painel da aplicação independente do módulo em que o utilizador está a trabalhar. Este botão tem um mecanismo equivalente a um *On/Off*, isto é, quando o utilizador carrega pela primeira vez, se existirem paredes seleccionadas e obras colocadas, as distâncias são calculadas e desenhadas, quando o utilizador voltar a carregar no botão, essa informação desaparecerá da cena.

São criadas tantas instâncias da classe *CalcDistances* quantas as paredes com obras associadas existam, ou seja, se existirem 10 paredes seleccionadas, 4 delas com obras, serão criadas 4 instâncias desta classe.

Cada instância criada irá chamar os métodos *GetFloorDistances* e *GetRightDistances*, isto é equivalente a dizer que, para cada obra de cada parede, irão ser calculadas e desenhadas as distâncias até ao chão e até à extremidade direita.

O método *GetFloorDistances* começa por ir buscar os nomes das obras que estão, de momento, associadas à parede. Apartir do(s) nome(s) recebido(s), vamos buscar as informações dos 4 pontos (cantos) da obra, que contém as posições (x,y,z) de cada ponto. Foi feito um método que irá determinar qual dos 4 pontos tem a coordenada y mais baixa, isto é, qual dos 4 pontos representa o ponto mais baixo da obra relativamente à parede. Com base no valor y desse ponto é então calculada a distância, através da diferença entre este valor y do ponto mais baixo da obra com o argumento *floor*, que representa o valor y do ponto mais baixo da parede, isto é, o chão.

Esta informação é passada para o método *PrintArrows*, que é o método responsável pelo desenho da informação na cena. Os argumentos, isto é, a informação que é passada para este método é a distância calculada, valor que será usado para definir o comprimento do traço, o ponto apartir do qual será desenhado o traço e um número inteiro, que informa se aquela distância é relativa ao chão ou à extremidade direita da parede, pois ambos os casos são distintos, nomeadamente na forma como se desenham as informações na cena.

O método *GetRightDistances* é praticamente igual ao *GetFloorDistances*, com a diferença de que é necessário ter em conta o vector normal da parede para a determinação de qual ponto se irá passar para o *PrintArrows*.

Portanto, resumindo, a ideia é determinar quais as paredes que têm obras, e sobre elas calcular e desenhar a distância de cada obra relativamente ao chão e à sua extremidade direita.

Utilizamos então uma cena simples para mostrar o aspecto da cena com o botão das distâncias activo:

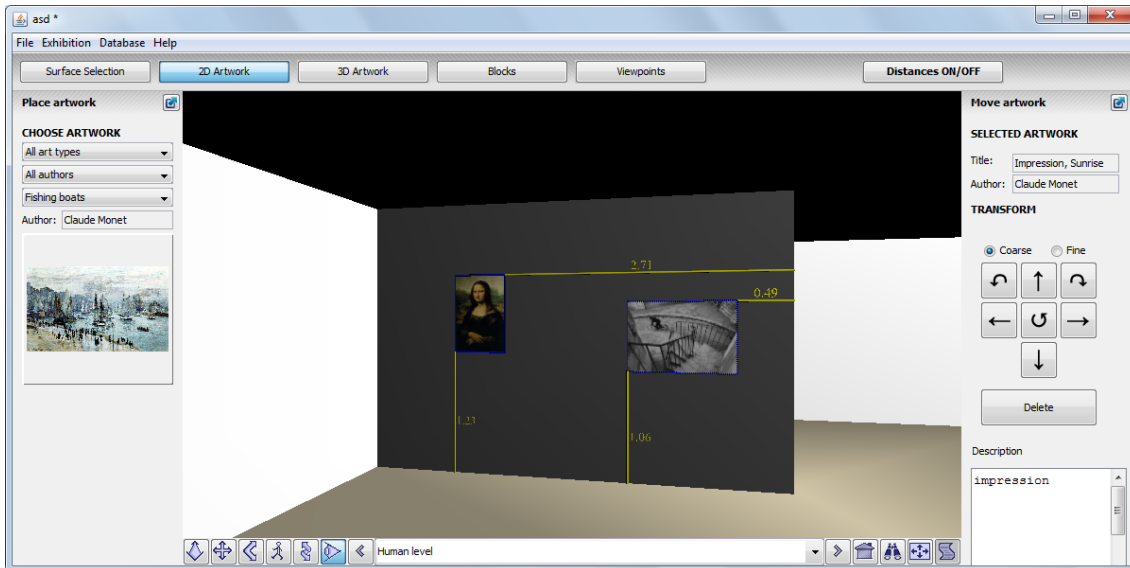



**Figura 4.12 - Exemplo de marcação de distâncias.**

As distâncias estão marcadas na cena a amarelo, pois é essa a côr que está definida para o seu desenho. São representados os traços e o valor da distância, em metros.

Uma forma alternativa para visualizar os valores das distâncias seria marcar as distâncias entre as obras em vez de desenhar a distância de cada obra à extremidade direita da parede.

Para isso, na altura, foi feito um método chamado *SortDistances* que ordenava as obras da menor para a maior distância relativamente à extremidade direita da parede. Depois destes valores estarem ordenados, era preparada a informação a passar para o *PrintArrows*. Comparativamente ao que foi realizado, isto é, ao que foi explicado atrás, apenas o comprimento do traço teria de ser calculado de outra forma, dado que o traço terá a mesma origem.

Para calcular a distância de uma obra para outra, era usado um array com as distâncias já ordenadas e esse array era percorrido de 2 em 2 casas, ou seja, para o desenho de um traço era sempre usada a obra imediatamente à sua direita, isto é, a obra que anterior no vector.

Imaginemos a seguinte situação: Temos uma parede com 3 obras. Segundo o método que descrevemos até agora, temos as distâncias das 3 obras relativamente à extremidade direita da parede, distâncias estas que estão ordenadas da mais próxima para a mais distante. Ora, o traço da obra mais perto terá, logicamente, de ser desenhado até à extremidade da parede, já o traço da 2ª obra terá de ser desenhado até ao ponto mais à esquerda da 1ª obra e por aí adiante.

Então, para o cálculo do comprimento do traço da 2ª obra terá de entrar a distância da 1ª até à parede mais a largura da 1ª obra.

Para facilitar o raciocínio do cálculo, mostramos a seguinte imagem:

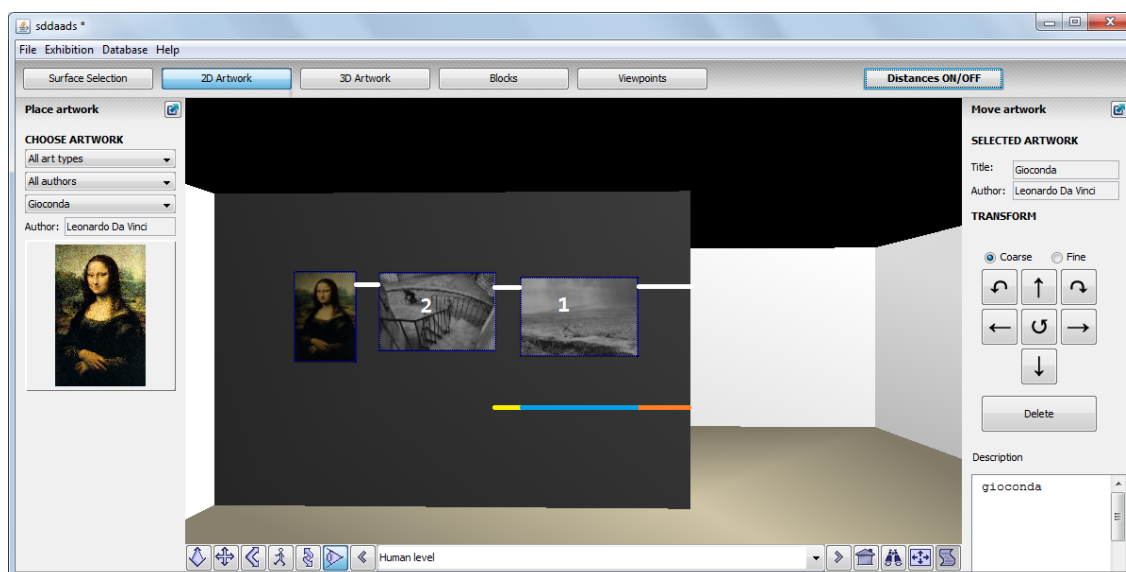



**Figura 4.13 - Exemplo explicativo da solução da distância entre obras.**

Os traços desenhados a branco corresponderiam ao aspecto que a cena teria realmente. O traço multicolor desenhado em baixo serve apenas para mostrar a ideia do cálculo.

Distancia (Obra2-Obra1) = d(Obra2) – d(Obra1) – w(Obra1). Isto é equivalente a:

d: distância obra-extremidade parede.

w: largura da obra.

Distancia (Obra2-Obra1) = comprimento total do traço multicolor – traço laranja – traço azul.

O resultante é o traço amarelo, equivalente à distância entre a 1ª e 2ª obra mais perto da extremidade direita da parede.

Mas isto poderia causar dificuldades na interpretação da informação desenhada em alguns casos, pois se as obras não estivessem alinhadas horizontamente o traço desenhado não coincidiria com as duas obras. Essa situação é exemplificada a seguir, na figura 4.14:

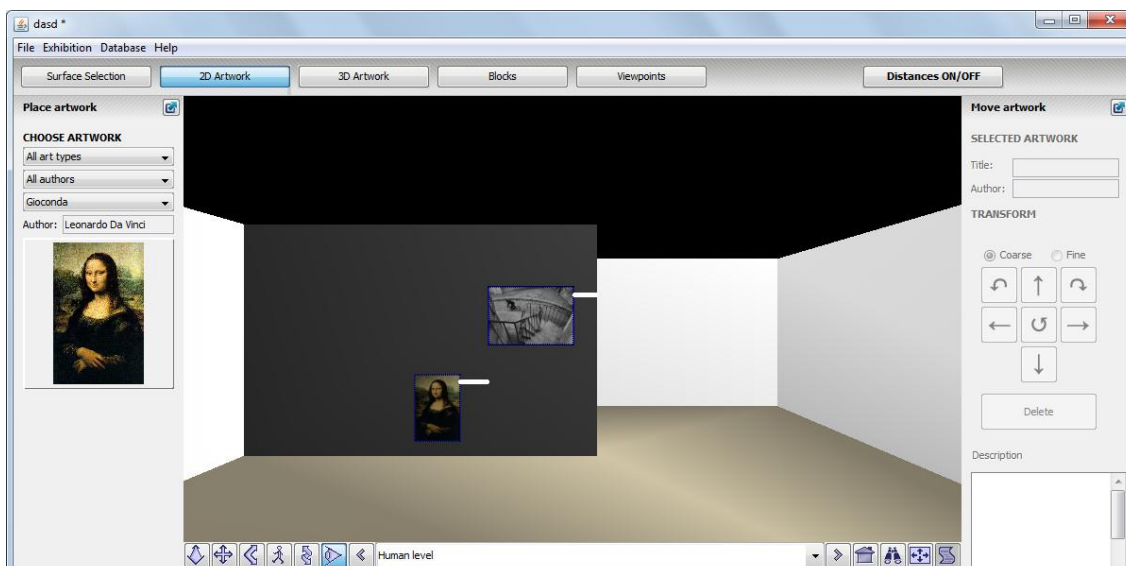



**Figura 4.14 - Exemplo se situação potencialmente confusa.**

Os traços a branco na cena junto às obras representam as distâncias tal como foram pensadas inicialmente.

Esta situação poderia causar alguma confusão ao utilizador, não tanto neste exemplo, mas com várias obras na parede, daí termos optado por utilizar directamente as referências para a extremidade direita da parede.

Inicialmente, e durante muito tempo, as informações sobre as obras e paredes estavam armazenadas num ficheiro de texto, que era editado conforme havia alterações na cena ou eram requeridas as informações para desenhar as distâncias na cena.

A estrutura do ficheiro era a seguinte:

```
Wall:id;normal;chao,dir;obra1,obra2

(...)

Obra1: ponto1,ponto2,ponto3,ponto4

(...)
```

Todo o trabalho de manutenção da integridade desse ficheiro estava assegurado pela classe *FileManipulator*. As principais razões pelas quais mudámos de planos relativamente a esta forma de armazenamento de informação foram o facto de, em alguns casos, alterações consecutivas muito rápidas acabarem por causar problemas de acesso ao ficheiro, isto é, a aplicação ter de fazer uma alteração no ficheiro enquanto a anterior ainda não está acabada, isto originou, por vezes, problemas de acesso para escrita. Juntando a isto, o facto de o desenho das distâncias ser dependente de um ficheiro de texto poderia causar problemas se, por algum acaso, a integridade do

ficheiro fosse comprometida. Finalmente, outro dos inconvenientes era a necessidade de termos sempre 3 ficheiros para cada exposição guardada, o ficheiro de estado, base de dados e distâncias.

Para concluir, a classe *Arts* descrita nesta secção, representa qualquer objecto que possa ser colocado na cena, nomeadamente as obras 2D, obras 3D e as Divisões.

Os 4 pontos que esta classe tem como argumento, no caso das obras 2D, correspondem aos 4 cantos do quadro/tapeçaria, já no caso das obras 3D e Divisões, os 4 pontos são os que têm contacto com a superície onde estão colocados, os outros 4 não entram nos cálculos.

Com esta informação estamos a dizer que também é possível calcular as posições das divisões colocadas no espaco físico da exposição.

## 4.4 Vitrines

Outra das funcionalidades introduzidas nesta nossa aplicação foi a possibilidade e colocar vitrines sobre as obras.

Para a realização deste *upgrade* socorremo-nos de um módulo já existente, o módulo *Division*, que tem como objectivo definir divisões temporárias no espaço físico da exposição, por exemplo, plintos.

Aliás, esta funcionalidade encontra-se disponível neste módulo dado que uma vitrine poderá ser considerada uma divisão extra.

Para que esta funcionalidade pudesse ser usada, definimos, para além das dimensões do bloco e a sua côr, a transparência. O utilizador poderá alterar a transparência de um bloco socorrendo-se de um *Jslider* que varia de 0 até 1. O 0 (zero) representa ausência de transparência, ou seja, opacidade; o valor 1 representa total transparência.

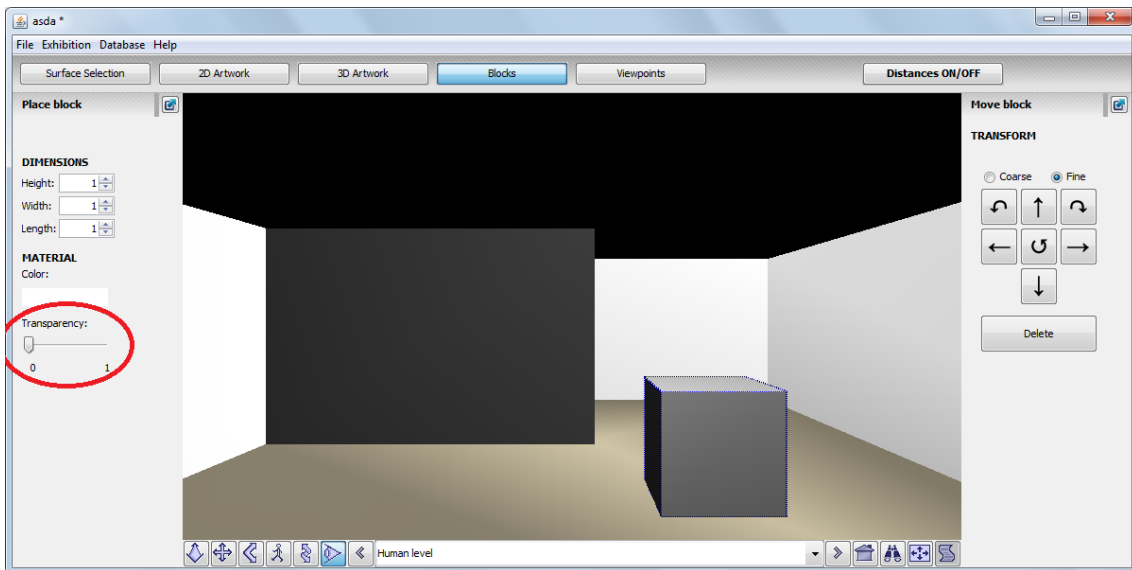



**Figura 4.15 - Exemplo de colocação de bloco opaco.**

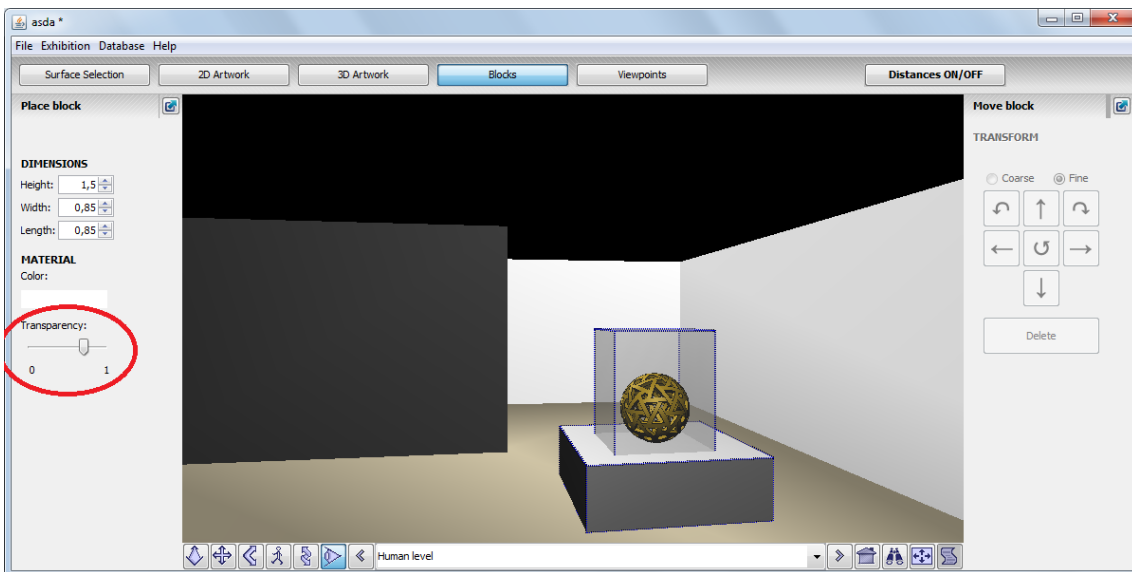



**Figura 4.16 - Exemplo da colocação de um objecto transparente (vitrine).**

## 4.5 Representação simplificada de obras 3D

A aplicação Virtual Exhibition Builder ja contemplava a criação de representação simplificada para obras 3D. Esta representação correspondia a uma primitiva box cujas dimensões dependiam do objecto original. Explicitamente eram indicadas as dimensões da *bounding box* que limita o objecto: comprimento, largura e altura. Esta box tinha associada uma imagem que era aplicada como uma textura em todas as faces visíveis.

O que nós fizemos foi permitir ao utilizador aplicar uma imagem, como textura, a cada face desse objecto.

Para o fazer, tivemos de alterar a geometria considerada anteriormente

Porque se tratava de uma geometria pré-definida, e era por isso impossível tratar cada face de forma diferenciada sendo o objecto considerado apenas 1 cubo, e não um conjunto de 6 faces, tivemos de re-criar o objecto.

O que fizemos foi criar 6 IndexedFaceSet para representar as 6 faces, e atribuimos uma textura diferente a cada uma, ou seja, na prática, era criada uma shape com uma *box* como geometria, agora são criadas 6 shapes, cada uma sendo um IndexedFaceSet.

O método que faz a construção do objecto tem como tipo de retorno um ObjectNode, cujo objecto era definido por um tuplo de 3 elementos, um X3Dnode (uma shape), e as coordenadas da *bounding box* (upper e lower).

Ora, dado que o objecto retornado do método que constrói o cubo com as várias texturas terá de retornar um conjunto de shapes, o construtor da classe ObjectNode foi alterado para receber um conjunto, neste caso escolhemos uma ArrayList de X3DNode.

Esta alteração, possibilidade de atribuir várias imagens a um objecto, não teve qualquer alteração na estrutura da base de dados, o campo onde ficava o caminho da (única) imagem do cubo que estava antes implementado é usado, mas usando vários caminhos separados por ";" (ponto e vírgula).

Se o utilizador associar apenas uma imagem ao objecto, essa imagem irá ser repetida por todas as faces, gerando um resultado equivalente ao que gerava antes desta alteração, mas se o utilizador introduzir mais do que uma imagem, as imagens serão colocadas nas devidas faces, ficando sem imagem as faces eventualmente sem nenhuma associada.

A associação das imagens às faces do objecto é feita através da inserção de obras 3D, na interface desenvolvida para esse fim, descrita na secção 4.2. A seguir, na figura 4.17, mostramos um exemplo de uma obra inserido com baixo nível de detalhe

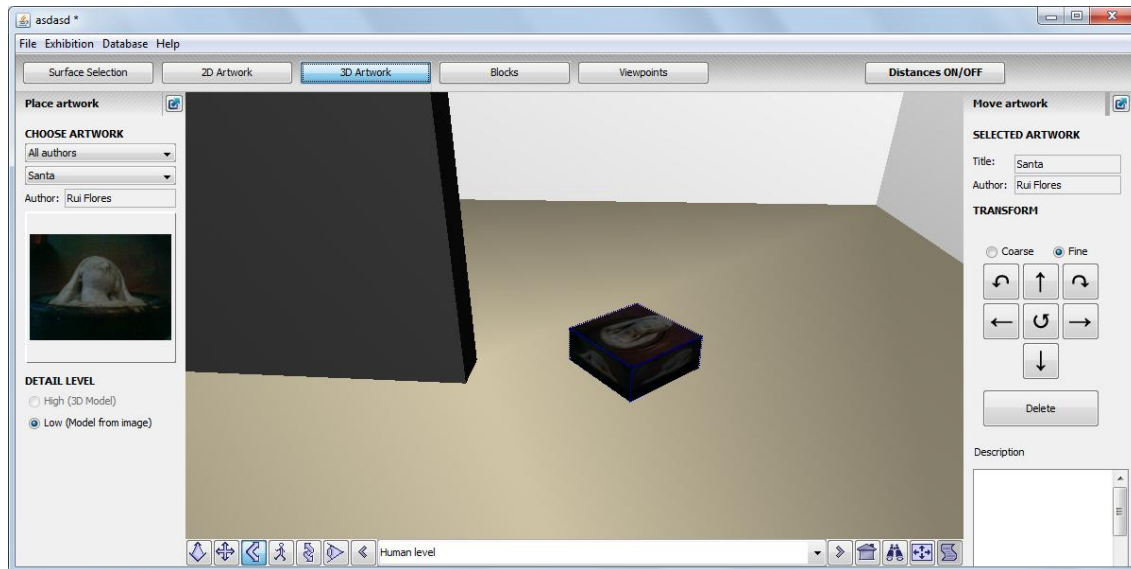



**Figura 4.17 - Exemplo de obra tridimensional com baixo nível de detalhe.**

## 4.6 Geração automática de Viewpoints

Para facilitar a visualização e recolha das distâncias das obras, achamos conveniente elaborar uma geração automática de *viewpoints*. Desta forma, quando a exposição estiver montada, o utilizador poderá recolher facilmente a informação das distâncias recorrendo ao módulo *Viewpoints*.

Quando o utilizador coloca uma obra na parede, é criado automaticamente um *viewpoint* dessa parede no módulo *Viewpoints*. Isto quer dizer que, todas as paredes com obras, terão um viewpoint definido. Quando o utilizador desejar, seleccionará o módulo *Viewpoints* e poderá navegar através deles.

Para criar um viewpoint é necessário dar-lhe um nome, uma localização e uma orientação. O nome é gerado automaticamente com o auxílio de uma variável incrementável, gerando "Wall i", com o i inicializado a zero e incrementado a cada criação de um *viewpoint*.

Quanto à localização, essa é dada pelo atributo *MidPt* da classe *Wall*, isto é, o ponto central da parede, informação ideal para o posicionamento do *viewpoint*, a orientação é obtida dependendo do vector normal da parede, isto é, o seu posicionamento relativamente à cena.

## 4.7 Ficheiro de configurações

Foi criado um ficheiro de configurações que servirá para o utilizador definir alguns valores por omissão da aplicação. Os valores que estão definidos são a pasta onde as bases de dados criadas na aplicação ficarão por omissão (pasta databases), a pasta onde ficarão, por omissão, as obras adicionadas na base de dados e finalmente a cor que o utilizador deseja que seja usada para a representação das distâncias na cena.

Este ficheiro tem uma estrutura baseada no XML, para facilitar a sua construção e leitura através do JAVA e também para ser facilmente perceptível (com o auxílio dos comentários escritos no próprio ficheiro).

O tipo de valor para a definição da pasta por omissão das bases de dados e seus recursos são os caminhos destas, por exemplo "C:/Rui/Museus/databases". O tipo de informação introduzida para a cor da representação das distâncias na cena é um tuplo de inteiros, correspondente ao sistema de cores RGB.

Um exemplo de um ficheiro de configuração:

```
<config>
<info>
<ColorDistanceMarks>1,1,0</ColorDistanceMarks>
<Resources_path>resources</Resources_path>
<Databases_path>databases</Databases_path>
</info>
<!--Tag ColorDistanceMarks - Represent the color of the distances marks.-->
<!--Tag Resources_path - Represent the default path of the resources added to Databases.-->
<!--Tag Databases_path - Represent the default path of the created databases.-->
<!--Note that the default paths are included in the Aplication root place.-->
<!--Some color representation examples. (default white)-->
<!--Red:1,0,0-->
<!--Yellow:1,1,0-->
<!--Blue:0,0,1-->
<!--Green:0,1,0-->
```

```
<!--White:1,1,1-->

<!--Black:0,0,0-->

</config>
```

Segundo este ficheiro de configuração, a cor para a representação das distâncias na cena é amarela, a directoria onde ficam os ficheiros das bases de dados criadas na ferramenta que desenvolvemos é a “databases”, na pasta raíz da aplicação, finalmente, a directoria onde ficam armazenados os recursos adicionados na base de dados é a “Resources”, também na pasta raíz do projecto

A informação colocada nas tags *ColorDistanceMarks, Resources_path e Databases_path* são carregadas aquando da execução da aplicação. Se o ficheiro for acidentalmente eliminado, um novo será criado na próxima execução da aplicação, com valores pré-definidos.

# Sumário

Neste capítulo apresentámos as extensões efectuadas à aplicação Virtual Exhibition Builder. Os diagramas de classe da aplicação encontram-se no Apêndice C.

# Capítulo 5 Discussão e trabalho futuro

Consideramos que, neste nosso projecto, conseguimos melhorar substancialmente a aplicação que estendemos, pois acrescentámos duas novas funcionalidades importantes no âmbito deste projecto, que foi a possibilidade de associar a um mesmo espaço físico diferentes colecções de arte, através da ligação a diferentes repositórios e a criação de uma interface amigável para o utilizador para a criação e actualização desses repositórios.

Tendo em conta esta alteração, alterámos a salvaguarda das exposições, isto é, a opção *Save As*. Esta opção guarda, não só o ficheiro de estado da exposição, como também a base de dados associada a esta, na mesma directoria. Para não existir uma duplicação exagerada de repositórios, criámos outro tipo se salvaguarda de exposições, que guarda apenas o ficheiro de estado, com um nome diferente. Isto permite ter várias exposições associadas a um mesmo repositório, sem ter de o duplicar.

Outro dos aspectos mais importantes era a criação de um mecanismo que mostrasse informações sobre as distâncias das obras relativamente às paredes. Para evitar confusão na interpretação da informação mostrada na cena, optámos por mostrar as distâncias das obras relativamente à extremidade direita da parede e ao chão.

Para facilitar a visualização da informação sobre distâncias criámos um mecanismo que cria *viewpoints* automaticamente, para as superfícies com obras, isto é, para cada superfície com obras, existirá, no módulo *Viewpoints*, um *viewpoint* definido.

Melhorámos também a representação de baixo detalhe das obras tridimensionais, sendo agora possível associar imagens individualmente as faces da bounding box que envolve o objecto tridimensional. Deste modo obtemos uma representação que aproxima o volume do objecto e que contém a sua imagem segundo diferentes pontos de vista.

Adicionámos também um novo parâmetro para as estruturas amovíveis, designadas no âmbito da ferramenta, divisões. Esse parâmetro foi a transparência, que, ao ser aplicada a um paralelepípedo, simula uma vitrine.

Finalmente, criámos um ficheiro de configurações que permite ao utilizador alterar alguns parâmetros por omissão da aplicação. Os parâmetros considerados foram a directoria onde serão armazenadas os repositórios de dados criados pela aplicação, a

directoria onde serão armazenadas as imagens ou ficheiros X3D que representam as obras de arte e a cor que o utilizador deseja utilizar para a visualização das informações das distâncias das obras na cena.

Como trabalho futuro pode estender-se esta aplicação de modo a mudar dinâmicamente a cor das superficies, a reproduzir música ambiente, a representar objectos suspensos e a reproduzir apresentações vídeo nas superfícies.

Como trabalho futuro, podem considerar-se também adaptações no módulo de edição que torne possível a movimentação de qualquer obra sem a necessidade de navegar nos botões de selecção dos módulos.

Consideramos também importante a realização de testes de usabilidade com utilizadores do público alvo, isto é, os técnicos de museus, para averiguar se as funcionalidades da aplicação satisfazem as suas necessidades.

A complexidade do tratamento das cenas tridimensionais em X3D foi um obstáculo na implementação de algumas tarefas, tais como a visualização de informações das distâncias das obras relativamente às superfícies e a representação simplificada de obras tridimensionais com várias texturas.

# Apêndice A – Guia de Utilização da Aplicação Java/X3D

Seguem as instruções que devem ser seguidas para a utilização da aplicação estendida no âmbito deste projecto, isto é, a aplicação Java/X3D.

Para utilizar esta aplicação, será necessário o seu terminal ter um sistema operativo Windows e que tenha as seguintes aplicações instaladas:

- Java Virtual Machine versão 7 ou superior [jvm7].
- BS Contact [bsc].

A instalação da Java Virtual Machine (JVM) permite ao utilizador correr aplicações Java no seu terminal.

A instalação do BS Contact permite ao utilizador visualizar as exposições resultantes da execução da nossa aplicação, que são representadas por ficheiros X3D.

Copia a pasta do projecto para uma directoria do disco e está habilitado a utilizar a nossa aplicação.

A pasta do projecto inclui, entre outros, os ficheiros: DBManager, que corresponde ao ficheiro de execução do programa de criação e actualização de repositórios de obras de arte; e VirtualGallery, que corresponde ao ficheiro para executar a criação e edição de exposições virtuais.

# Apêndice B – Manual de Utilização do Protótipo WebGL

Seguem as instruções que devem ser seguidas para a utilização do protótipo desenvolvida em WebGL, no âmbito deste projecto.

Para a utilização deste protótipo, o utilizador terá de ter um web browser instalado, poderá optar por qualquer um menos o Internet Explorer, que não suporta a tecnologia WebGL. Os web browser recomendados para a visualização deste protótipo são o Google Chrome e o Mozilla Firefox.

Poderá consultar as informações relativas à compatibilidade com os web browsers, que estão presentes no capítulo 3.3.

Relativamente ao Google Chrome, dependendo da versão, poderão existir problemas no carregamento das imagens associadas às texturas presentes no protótipo. Para resolver este problema, caso surja, o utilizador terá de activar uma flag nas configurações do web browser. Uma maneira simples de o fazer é aceder às propriedades do atalho do ficheiro executável do Google Chrome, e na tab "shortcut" terá o caminho do exectutável à frente do atributo "target". No fim desse caminho, adicione o seguinte: --allow-file-access-from-files.

Por exemplo, o utilizador terá o caminho "c:// ... /chrome.exe", apenas terá de adicionar "--allow-file-access-from-files", ficando "c:// ... /chrome.exe --allow-file-access-from-files". Assim, o problema fica resolvido e será possível visualizar os objectos com as suas texturas.

Após isto feito, basta copiar a pasta com os recursos do protótipo e está habilitado a visualizá-lo.

A pasta desta aplicação contém o ficheiro html com a codificação do protótipo, uma directoria que contém as bibliotecas usadas e uma directoria que contém as imagens usadas como texturas.

# Apêndice C – Diagramas de Classe

Começamos por referir o diagrama de classes resumido, correspondente à figura 2.1.

Usamos este diagrama de classes resumido dado que não foram criados módulos, apenas se actualizaram os já existentes.

O Diagrama de classes geral seria muito grande e as suas relações não ficariam claras, por isso iremos descrever o diagrama de classes por pacote, procedendo à sua relação posteriormente.

De seguida, apresentamos uma breve descrição de cada pacote:

| *Pacote* | *Descrição* |
|---|---|
| **Core** | Conversão do modelo x3d e inicialização dos módulos. |
| **Core.Module** | Operações e definição de caraterísticas comuns dos diversos módulos. |
| **Core.Surface** | Detecção e Selecção das superfícies do modelo. |
| **Core.Surface.Filters** | Definição de filtros de selecção de superfícies. |
| **Database** | Conecção à base de dados. |
| **GUI** | Graphic User Interface da aplicação. |
| **Modules.art2d** | Representação do módulo 2D. |
| **Modules.art3d** | Representação do módulo 3D. |
| **Modules.division** | Representação do módulo da divisões. |
| **Modules.viewpoints** | Representação do módulo dos viewpoints. |
| **Util** | Operações úteis à aplicação. |

**Tabela C.1 – Descrição geral dos pacotes da aplicação.**

Apresentamos de seguida a descrição das classes, por pacote:

| *Classe* | *Descrição* |
|---|---|
| Converter | Converte a cena do ficheiro original para a cena que é usada na aplicação (normalização). |
| Gallery | Inicialização da aplicação, com os dados do modelo e base de dados. |
| ModuleLoader | Inicialização dos módulos. |
| ProjectConfiguration | Regista as características de uma nova exposição. |
| ProjectManager | Regista e carrega o estado de cada módulo aquando de um *save* ou *open*, respectivamente. |

**Tabela C.2 – Descrição das classes do pacote core**

| *Classe* | *Descrição* |
|---|---|
| AbstractSceneObject | Representação abstrata dos objectos colocados na cena. |
| ObjectBoundingBox | Obtenção das bounding boxes dos objectos colocados na cena. |
| ObjectPlacingMode | Representação abstrata das operações sobre os objectos. |
| OperationModule | Gere as transições de módulo. |

**Tabela C.3 – Descrição das classes do pacote core.module**

| *Classe* | *Descrição* |
|---|---|
| AutoPicker | Aplicação dos filtros de superfície. |
| FlatShape | Representa uma superfície. |
| ManualPicker | Selecciona uma superfície com um clique. |
| Surface | Operações sobre a superfície. |
| SurfaceDetector | Detecta superfícies na cena original. |
| SurfaceEvent | Evento lançado aquando de uma acção sobre uma superfície seleccionada ou detectada |
| SurfaceGroup | Operações de conjunto (de superfícies). |
| SurfaceModule | Prepara toda a informação do módulo Surface. |

| | |
|---|---|
| SurfaceModuleGUI | GraphicalUserInterface do módulo. |
| SurfaceSelectionInterface | Representa todas as acções de selecção de superfícies. |
| Triangle | Representação dos triângulos dos quais as superfícies são compostas. |

**Tabela C.4 – Descrição das classes do pacote core.surface**

| *Classe* | *Descrição* |
|---|---|
| Areafilter | Descreve o filtro de área de superfície. |
| DisjuntionFilter | Descreve o filtro de normais básico. |
| Filter | Descrição genérica de um filtro. |
| FilterSet | Passa os filtros existentes para a aplicação. |
| NormalFilter | Descreve o filtro de normais avançado. |

**Tabela C.5 – Descrição das classes do pacote core.surface.filters**

| *Classe* | *Descrição* |
|---|---|
| AuthorInfo | Contém informação sobre autores de obras. |
| CreateDB | Cria uma base de dados. |
| DBConnection | Estabelece uma ligação a uma base de dados. |

**Tabela C.6 – Descrição das classes do pacote database**

| *Classe* | *Descrição* |
|---|---|
| CreateDatabase | Janela inicial da aplicação DBManager. |
| EditDatabase2D | Janela de actualização de obras de arte 2d |
| EditDatabase3D | Janela de actualização de obras de arte 3d. |
| Fetcher | Comunicação com a base de dados. |
| GUIController | Carregamento da interface geral da aplicação. |
| Insert2dData | Janela de inserção de obras de arte 2d. |
| Insert3dData | Janela de inserção de obras de arte 3d. |

| | |
|---|---|
| ModeChooser | Classe responsável pela transição de módulos. |
| NewProjectWindow | Janela inicial da criação de uma exposição. |
| Transform | Painel com os botões de deslocação das obras. |
| VirtualGallery | Main class da aplicação. |

**Tabela C.7 – Descrição das classes do pacote gui.**

| *Classe* | *Descrição* |
|---|---|
| Art2Dmodule | Carrega e gere todo o módulo 2D. |
| Artwork2D | Representa um Objecto 2D. |
| Artwork2Dinfo | Contém informações da base de dados sobre um objecto 2D. |
| Fetcher | Trata das queries à base de dados necessárias para a pesquisa de obras para colcação. |
| ObjectPanel | Representa o GUI do painel direito do módulo. |
| PlacementPanel | Representa o GUI do painel esquerdo do módulo. |
| SelectArtEvent | Evento lançado aquando da selecção de uma obra 2D. |

**Tabela C.8 – Descrição das classes do pacote modules.art2d**

| *Classe* | *Descrição* |
|---|---|
| Art3Dmodule | Carrega e gere todo o módulo 3D. |
| Artwork3D | Representa um objecto 3D. |
| Artwork3Dinfo | Contém informações da base de dados sobre um objecto 3D. |
| ArtworkClumsy | Representação de baixo nível de detalhe de obra 3D. |
| ArtworkModel | Representação de alto nível de detalhe de obra 3D. |
| Fetcher | Tratamento das queries à base de dados necessárias para a pesquisa de obras para colcação. |
| ObjectPanel | Representação do painel direito do módulo 3D. |
| PlacementPanel | Representação do painel esquerdo do módulo 3D. |
| SelectArtEvent | Evento lançado aquando da selecção de uma obra 3D. |

**Tabela C.9 - Descrição das classes do pacote modules.art3d**

| *Classe* | *Descrição* |
|---|---|
| Division | Representação de uma divisão. |
| DivisionModule | Carregamento e manutenção de todo o módulo Division. |
| DivisionSelected | Evento lançado aquando de uma selecção de uma divisão. |
| ObjectPanel | Representação do painel direito do módulo Division. |
| PlacementPanel | Representação do painel esquerdo do módulo Division. |

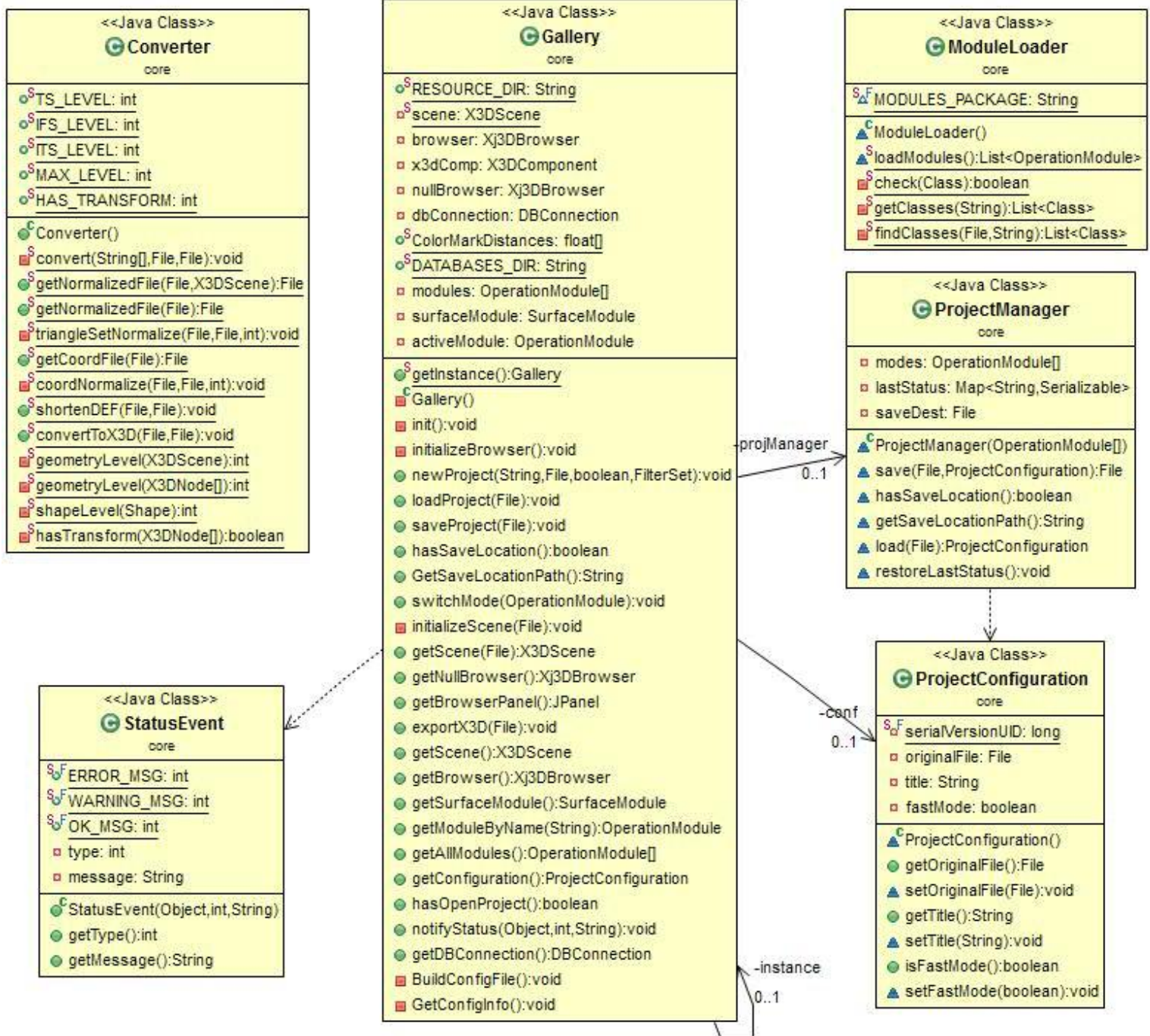
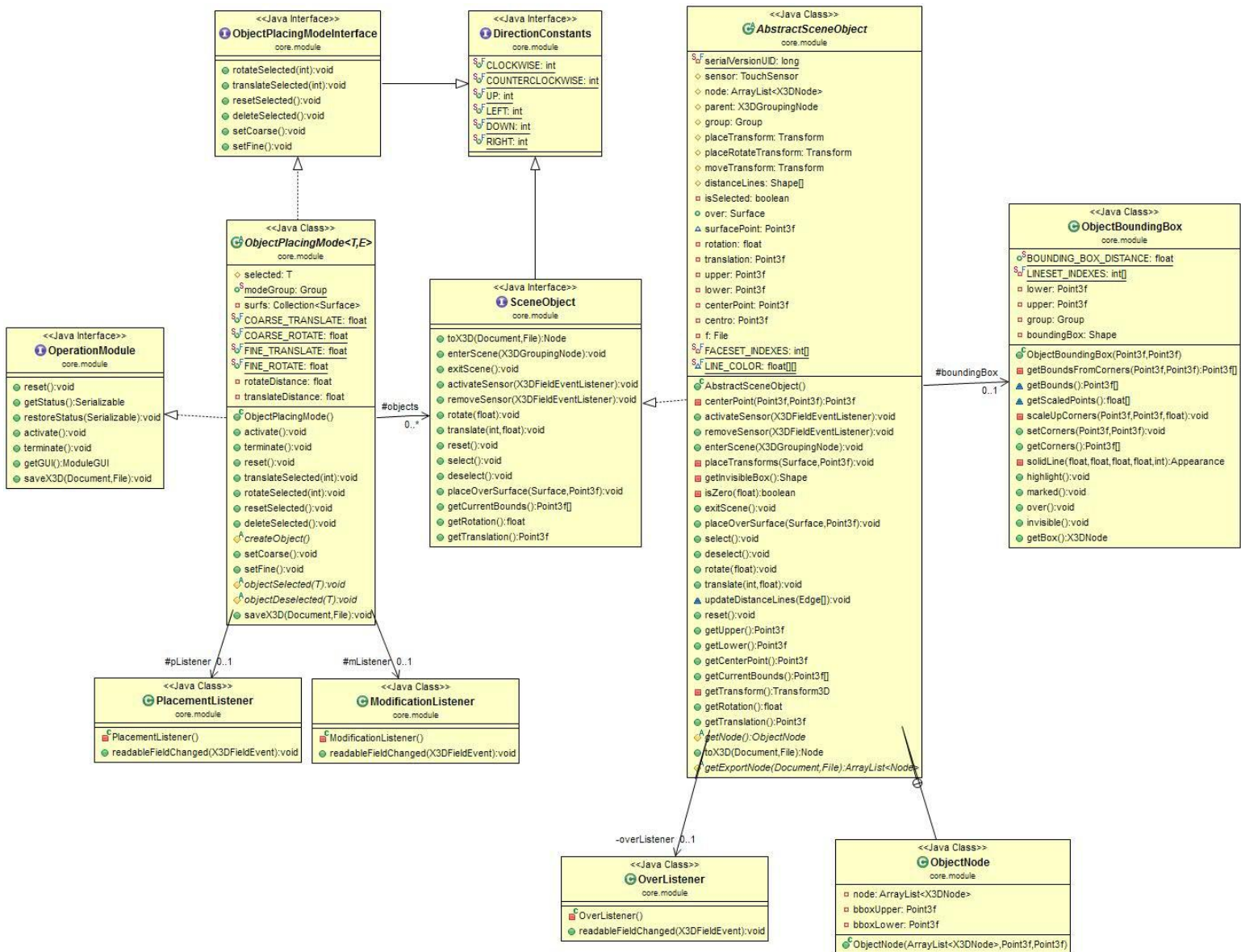
**Tabela C.10 - Descrição das classes do pacote modules.division**

| *Classe* | *Descrição* |
|---|---|
| SceneViewpoint | Representação de um viewpoint. |
| ViewpointGUI | GUI do módulo Viewpoint. |
| ViewpointModule | Carregamento e manutenção de todo o módulo Viewpoint. |

**Tabela C.11 - Descrição das classes do pacote modules.viewpoints**

| *Classes* | *Descrição* |
|---|---|
| Art | Representação da informação das obras colocadas nas superfícies. |
| CalcDistances | Cálculo e desenho das distâncias na cena. |
| CalcPositions | Cálculo das posições dos objectos colocados nas superfícies. |
| FileManipulator | Operações associadas a criação e manipulação de ficheiros. |
| Utils | Funções úteis de classes de outros pacotes. |
| Wall | Representação da informação das paredes seleccionadas. |

**Tabela C.12 - Descrição das classes do pacote util**

Apresentadas as classes de cada pacote, mostramos de seguida os diagramas de classe de cada um deles, pela mesma ordem com que foram apresentadas as suas classes:

**C.1 – Diagrama de classes do pacote core**

**C.2 – Diagrama de classes do pacote core.module**

**C.3 – Diagrama de classes do pacote core.surface**

<<Java Class>>
DisjunctionFilter
core.surface.filters

DisjunctionFilter()
addFilter(Filter):void
accepts(Surface):boolean
equals(Object):boolean
hashCode():int

-filters 0..1

<<Java Class>>
FilterSet
core.surface.filters

serialVersionUID: long

FilterSet()
passFilters(Surface):boolean
add(Filter):boolean

<<Java Class>>
*Filter*
core.surface.filters

Filter()
*accepts(Surface):boolean*
equals(Object):boolean
hashCode():int

<<Java Class>>
AreaFilter
core.surface.filters

minimumArea: float

AreaFilter(float)
accepts(Surface):boolean

<<Java Class>>
NormalFilter
core.surface.filters

HORIZONTAL: int
VERTICAL: int
OBLIQUE: int
ZERO: int
NON_ZERO: int
ALL: int
xRestriction: int
yRestriction: int
zRestriction: int

NormalFilter(int,int,int)
NormalFilter(int)
accepts(Surface):boolean
componentAccepted(int,float):boolean
hashCode():int
equals(Object):boolean

**C.4 – Diagrama de classes do pacote core.surface.filters**

<<Java Class>>
AuthorInfo
database

name: String
nationality: String
id: int

AuthorInfo()
setName(String):void
setNationality(String):void
setID(int):void
getName():String
getNationality():String
getID():int

<<Java Class>>
DBConnection
database

conn: Connection
stm: Statement
dbpath: String
db_dir: String

DBConnection(String)
setDbdir():void
getDbdir():String
getDbpath():String
update(String):void
query(String):ResultSet

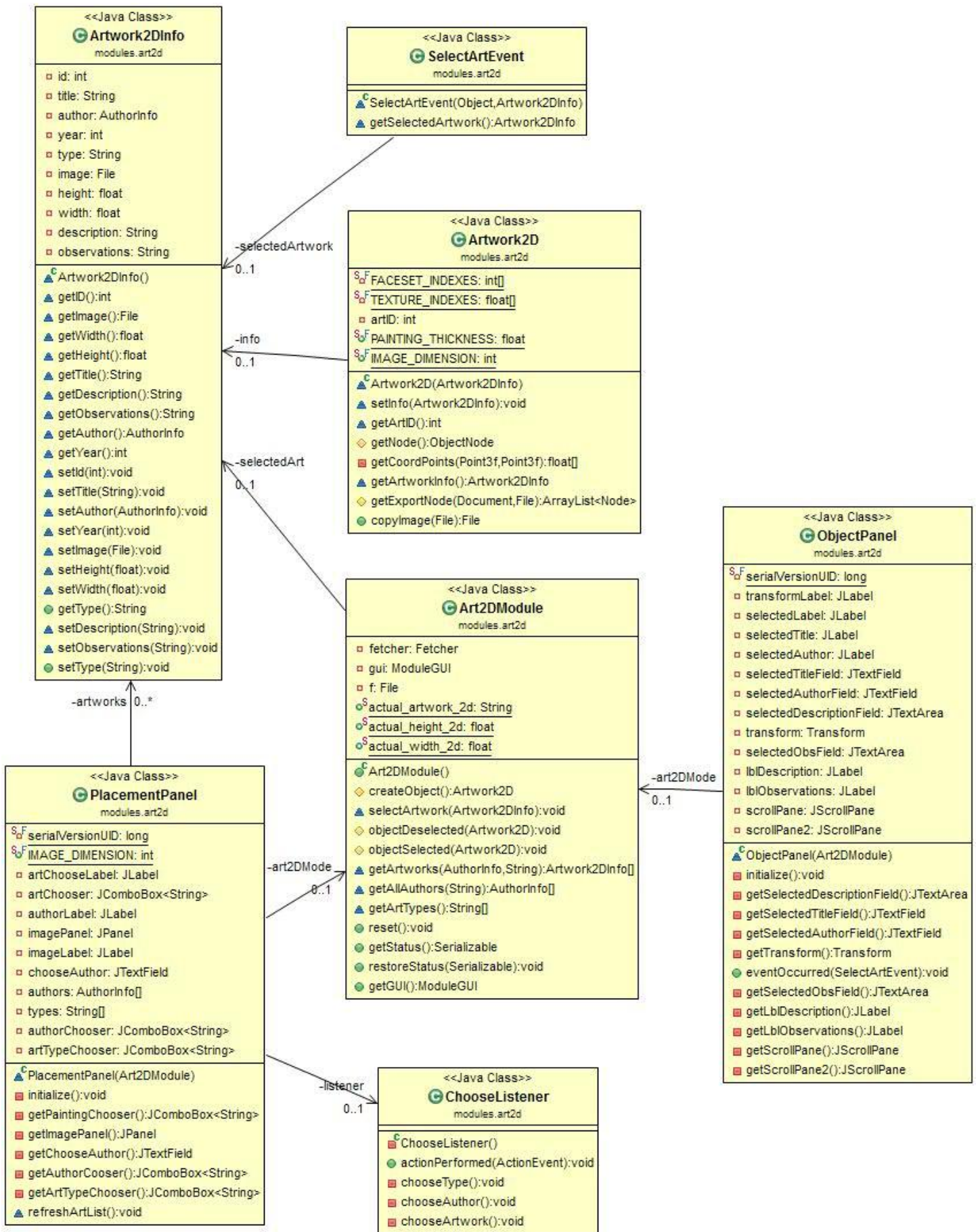
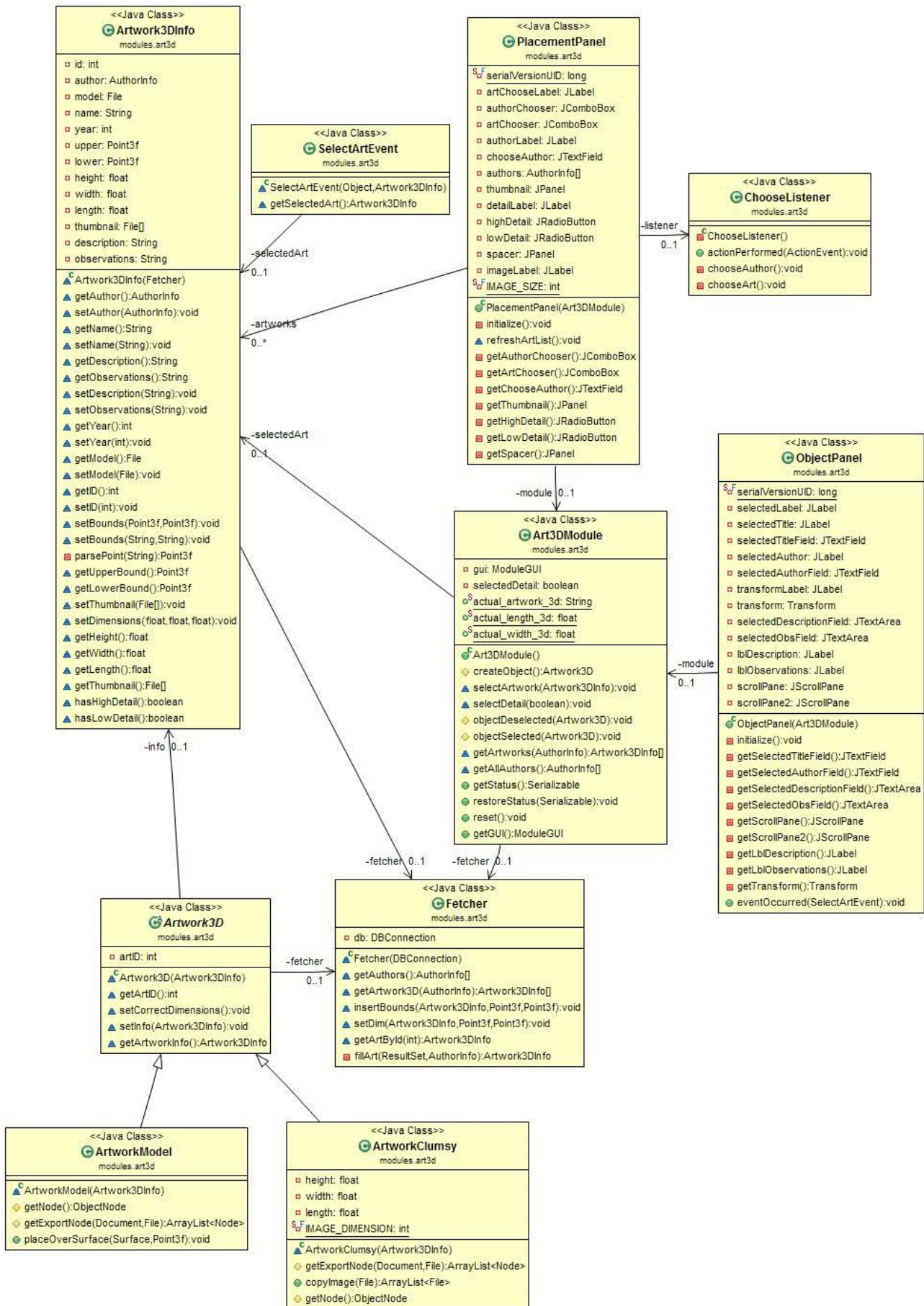
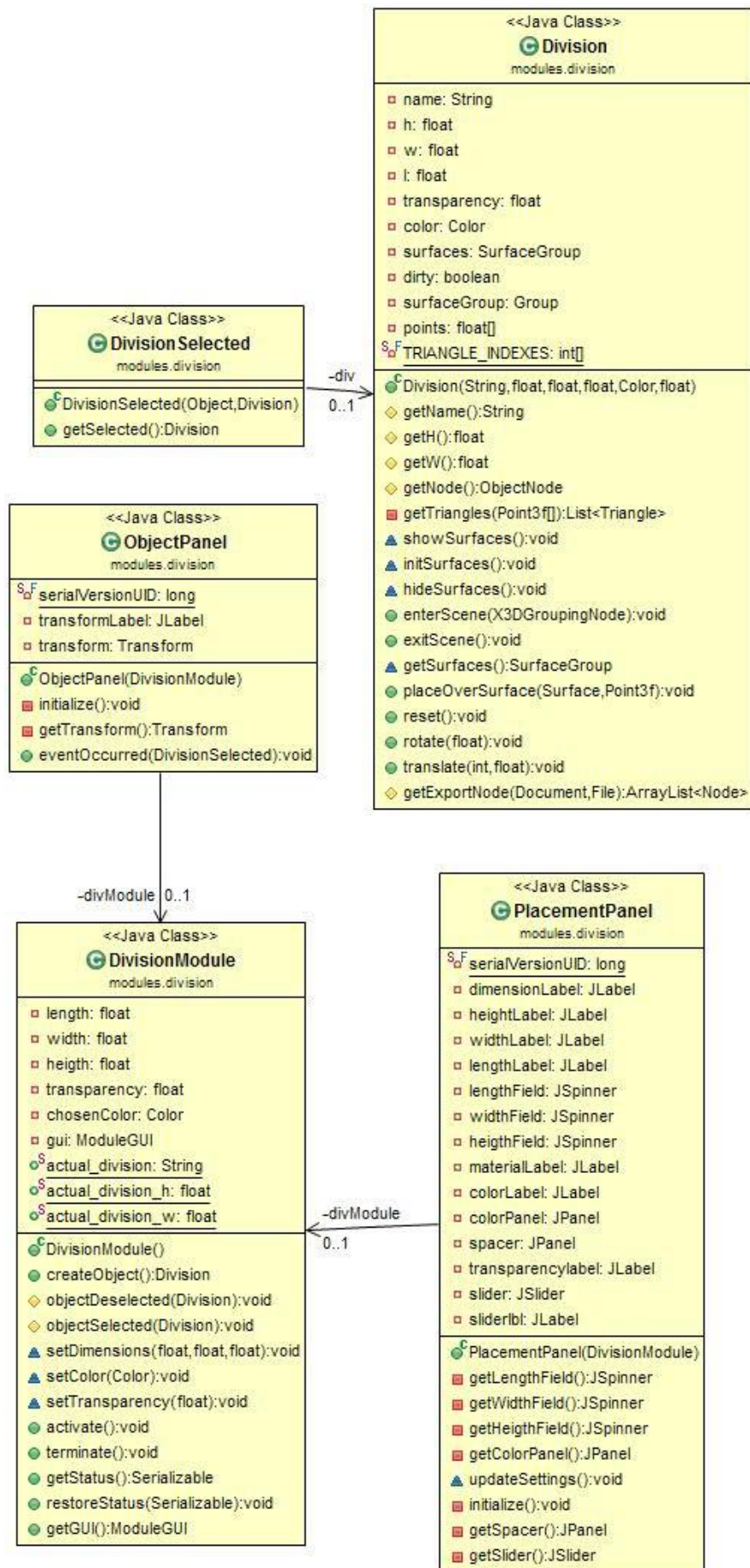
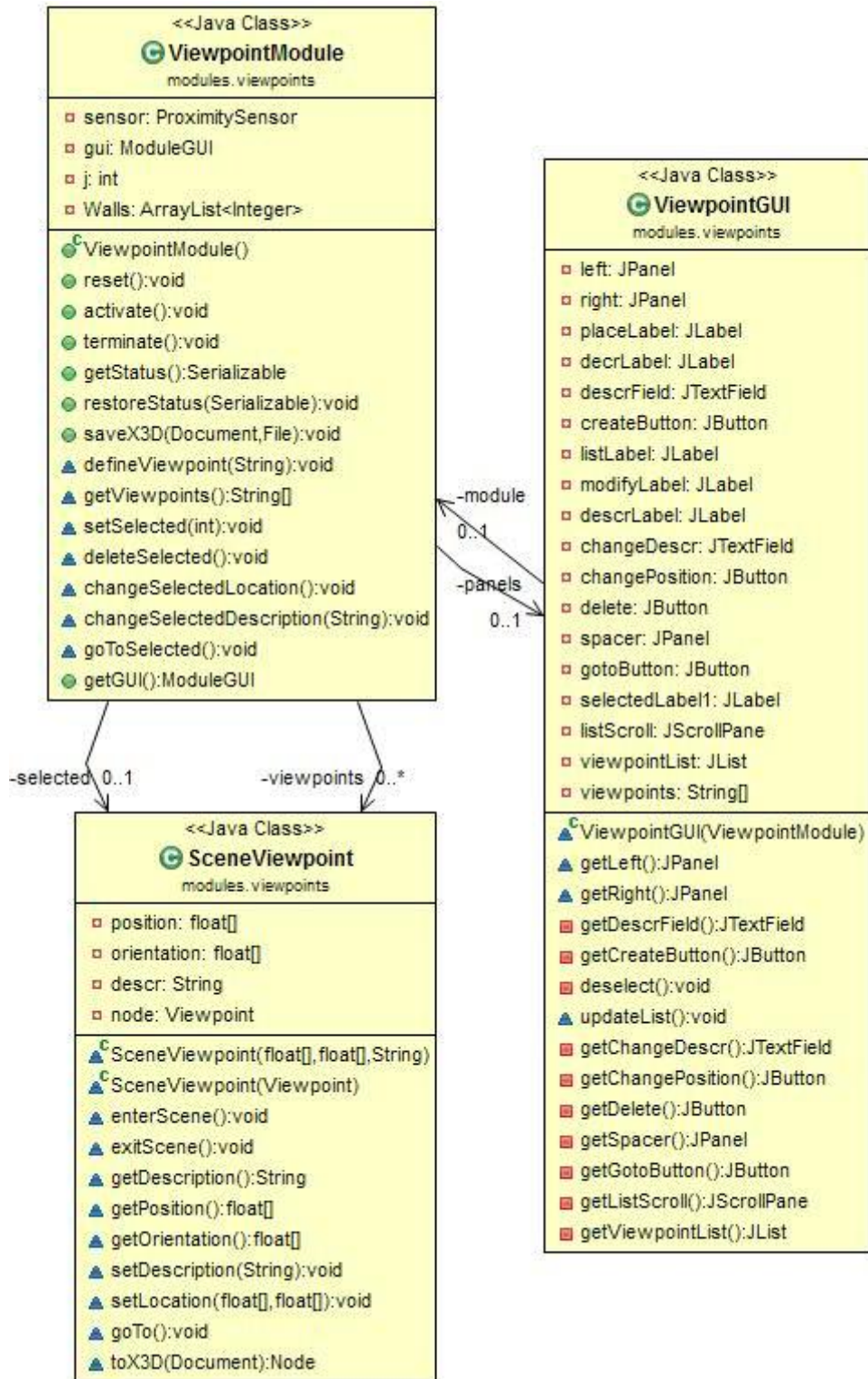
~dbc 0..1

<<Java Class>>
CreateDB
database

dbName: String

CreateDB(String)
GetDbName():String
run():void

**C.5 – Diagrama de classes do pacote database**

**C.6 - Diagrama de classes do pacote gui**

**C.7 - Diagrama de classes do pacote modules.art2d**

**C.8 - Diagrama de classes do pacote modules.art3d**

**C.9 - Diagrama de classes do pacote modules.division**

**C.10 - Diagrama de classes do pacote modules.viewpoints**

Apresentadas que estão as classes e os diagramas de classe de cada pacote (dispensamos o diagrama de classes do pacote util, dado que não é relevante e não ha relação alguma entre as suas classes), vamos agora proceder à descrição das relações entre eles:

O pacote “core” está relacionado com os pacotes “core.module” e “core.surface”, sendo o 1º (core.module) responsável pela fluidez da interacção do utilizador com os módulos e a sua transição e o 2º é o responsável pela detecção de superfícies através da uniformização da geometria da cena, descrita na secção 2.6.1 e a gestão do módulo Surface.

Ora, sabendo que o pacote “core.module” contém as representações abstratas dos objectos que se podem colocar na cena e uma representação abstrata de colocação de objecto na cena, isso relaciona-o com os pacotes “modules.art2d”, “modules.art3d”, “modules.division” e com “modules.viewpoints”, embora este último não represente um objecto que se coloque fisicamente na cena.

O pacote “gui” relaciona-se com o todos os pacotes, excepto com o “util”, “core.surface.filters” e “database”, que não têm qualquer interface gráfica associada. Este pacote é responsável também pela permuta de interfaces gráficas aquando de mudanças de módulo.

O pacote “database” está relacionado com os pacotes “core”, “gui”, “modules.art2d” e “modules.art3d”, dado que todos estes necessitam de aceder à base de dados. Individualizando, o pacote “core”, aquando do processo de criação ou carregamento de uma exposição, procede a uma ligação à base de dados fornecida pelo utilizador por forma a preencher as informações relativas às obras nos seus respectivos módulos. O pacote gui contém as classes que representam a aplicação para criação e actualização de bases de dados, descrita na secção 4.1, que cria/acede a uma base de dados e actualiza-a.

O pacote “core.surface.filters” apenas representa os filtros associadas às superfícies, estando apenas relacionado com o pacote “core.surfaces”.

O pacote “division” está relacionado com o pacote “core.surface”, dado que quando é criada uma divisão, esta é imediatamente transformada numa superfície elegível pela colocação de obras.

# Apêndice D – Mapa de Gantt

Segue o mapa de Gantt, referente ao planeamento do projecto desenvolvido.

Como já foi referido no Capítulo 5 – Discussão e Trabalho Futuro, a complexidade do tratamento das cenas tridimensionais da aplicação Virtual Exhibition Builder levou a que a concretização de algumas tarefas demorasse mais tempo que o previsto, tal como alguns problemas pessoais, que impediram durante um largo período, uma produtividade aceitável.

Devido a isto, o planeamento inicial divergiu do esperado e fez com que o trabalho se arrastasse durante mais tempo que o previsto.

GANTT project
Name
Begin date
End date
2011
2012
October
November
December
January
February
March
April
May
June
July
August
September
Adaptação do projecto 9/30/11 10/14/11
Adaptação às ferramentas do trabalho 9/30/11 10/14/11
Adaptação às tecnologias utilizadas no trabalho 9/30/11 10/14/11
Pesquisa sobre o estado da arte no âmbito do projecto 9/30/11 10/14/11
Pesquisa de bibliotecas WebGL 10/17/11 10/31/11
Implementação do protótipo em WebGL 11/1/11 11/30/11
Primitivas Gráficas 11/1/11 11/3/11
Navegação na cena 11/4/11 11/7/11
Sensores de toque 11/8/11 11/11/11
Fontes de Luz 11/14/11 11/17/11
Texturas 11/18/11 11/22/11
Interface de interacção com utilizador 11/23/11 11/25/11
Projecção de sombras 11/28/11 11/30/11
Escrita do Relatório Preliminar 12/1/11 1/13/12
Extensão Aplicação Java/X3D 1/16/12 8/15/12
Componente de criação e edição de bases de dados 1/16/12 2/15/12
Salvaguarda de Exposições 2/16/12 2/29/12
Visualização dos valores das distâncias 3/1/12 6/20/12
Geração automática de Viewpoints 6/21/12 6/25/12
Vitrines 6/26/12 6/29/12
Representação Low Detail Obras 3D 7/2/12 8/10/12
Ficheiro de configuração 8/13/12 8/15/12
Escrita do relatório Final 8/16/12 9/25/12

# Bibliografia

[blogWebGL] http://learningwebgl.com/blog/?page_id=2 (17-11-2011) Blog Learning WebGL.

[bsc] http://www.bitmanagement.com/en/download (24-09-2012) Página de download do BS Contact

[DAT.GUI] http://code.google.com/p/dat-gui/ (28-11-2011) A lightweight library for JavaScript

[Eclipse] http://www.eclipse.org/ (05-02-2011) Eclipse

[galBuilder] http://www.youtube.com/watch?v=grzdbdoFz1M (17-11-2011)

Browserbased 3D Gallery Builder.

[GLGE] http://www.glge.org/ (17-11-2011) WebGL for the lazy

[Gomes10] Jorge Carvalho Gomes, Maria Beatriz Carmo, Ana Paula Cláudio, "Construção Interactiva de Exposições Virtuais", INForum 2010 – Simpósio de Informática, Braga, 9-10 Setembro, 2010.

[Gomes11] Jorge Carvalho Gomes, Maria Beatriz Carmo, Ana Paula Cláudio: "Virtual Exhibition Builder", International Conference on Computer Graphics Theory and Applications, GRAPP, 2011.

[Hrk01] Hrk, S.: Virtual Art Gallery, CESCG, 2001.

[IntWebGL] http://gdd11-webgl.appspot.com/#27 (17-11-2011) Introduction to WebGL.

[jvm7] http://www.java.com/pt_BR/download/chrome.jsp?locale=pt_BR (24-09-2012) Página de download da JVM 7

[Katalabs] http://www.katalabs.com/blog/ (17-11-2011) Katalabs.

[NetBeans] http://netbeans.org/ (04-01-2012) NetBeans

[Patias08] Patias, P., Chrysantou, Y., Sylaiou, S., Georgiadis, Ch., Michail, D. M.,

[RTGomes11] Jorge Gomes, Maria Beatriz Carmo, Ana Paula Cláudio: Creating and Assembling Virtual Exhibitions from Existing X3D Models, Relatório Técnico DI-FCUL-TR-2011-4.

[Semião07] Pedro Semião , P. M., Carmo, M. B.: Galeria de Arte Virtual, Actas do 15º Encontro Português de Computação Gráfica, poster, 2007.

[Semião08] Pedro Semião, Maria Beatriz Carmo, Virtual Art Gallery Tool, Proceedings GRAPP, 2008.

[Sledz07] Daniel A. Sledz, Donald E. Coomes, Mathias Kolsch, Michael Thompson: "A dynamic three-dimensional network visualization Program for integration into cyberciege and other network visualization scenarios, Master Thesis, 2007.

[SpiderGL] http://spidergl.org/index.php (15-11-2011) SpiderGL 3D Graphics for next-generation WWW

[sqliteAbout] http://www.sqlite.org/about.html (21-08-2012) About SQLite.

[Three] https://github.com/mrdoob/three.js/ (19-11-2011) JavaScript 3D library

[visVirt3D] http://www.culturaonline.pt/MuseusMonumentos/Pages/visitasvirtuais.aspx (28-09-2011) Visitas virtuais 3D.

[webGLOpenGL] http://www.khronos.org/webgl/wiki/WebGL_and_OpenGL (17-11-2011) WebGL and OpenGL.

[wikiWebGL] http://www.khronos.org/webgl/wiki/Main_Page (17-11-2011) Wiki WebGL.

[Woj04] Wojciechowski, R., Walczack, K, White, M., Cellary, W.: Building Virtual and Augmented Reality Museum Exhibitions, Proceedings of the 9th International Conference on 3D Web Tecnology, 2004.

[Xampp] http://www.apachefriends.org/pt_br/xampp.html (04-01-2012) Apache Friends.